

SEPARATE, MEASURE AND CONQUER: FASTER ALGORITHMS FOR MAX 2-CSP AND COUNTING DOMINATING SETS

SERGE GASPERS AND GREGORY B. SORKIN

ABSTRACT. We show a method resulting in the improvement of several polynomial-space, exponential-time algorithms.

An instance of the problem MAX $(r, 2)$ -CSP, or simply MAX 2-CSP, is parametrized by the domain size r (often 2), the number of variables n (vertices in the constraint graph G), and the number of constraints m (edges in G). When G is cubic, and omitting sub-exponential terms here for clarity, we give an algorithm running in time $r^{(1/5)n} = r^{(2/15)m}$; the previous best was $r^{(1/4)n} = r^{(1/6)m}$. By known results, this improvement for the cubic case results in an algorithm running in time $r^{(9/50)m}$ for general instances; the previous best was $r^{(19/100)m}$. We show that the analysis of the earlier algorithm was tight: our improvement is in the algorithm, not just the analysis. The new algorithm, like the old, extends to Polynomial and Ring CSP (encompassing graph bisection, the Ising model, and counting). It is also the fastest polynomial-space algorithm for MAX CUT, for cubic and general instances.

We also give faster algorithms for #DOMINATING SET, counting the dominating sets of every cardinality $0, \dots, n$ for a graph G of order n . For cubic graphs, our algorithm runs in time $3^{(1/6)n}$; the previous best was $2^{(1/2)n}$. For general graphs, we give an unrelated algorithm running in time 1.5183^n ; the previous best was 1.5673^n .

The previous best algorithms for these problems all used local transformations and were analyzed by the “Measure and Conquer” method. Our new algorithms capitalize on the existence of small balanced separators for cubic graphs — a non-local property — and the ability to tailor the local algorithms always to “pivot” on a vertex in the separator. The new algorithms perform much as the old ones until the separator is empty, at which point they gain because the remaining vertices are split into two independent problem instances that can be solved recursively. It is likely that such algorithms can be effective for other problems too, and we present their design and analysis in a general framework.

1. INTRODUCTION

1.1. Background and intuition of the method. The use of graph separators for divide-and-conquer algorithms dates back at least to the 1970s [LT77]. For classes of instances with sub-linear separators, including those described by planar graphs, this often gives subexponential- or polynomial-time algorithms. It is natural to design a branching strategy that strives to disconnect an instance into components, even when no sublinear separators are known. While this has successfully been done experimentally (see, e.g., [BS06, DM07, FQ85, GLS00, LvB04]), we are not aware of worst-case analyses of branching algorithms that are based on linear sized separators.

Our algorithms exploit small separators, specifically, balanced separators of size about $n/6$ for any cubic graph of order n . The existence of such separators has been known since 2001 at least, they have been used in pathwidth-based dynamic programming algorithms using exponential time and exponential space, and it is natural to try to exploit them for algorithms running in exponential time but polynomial space.

Outlining the paper (a more detailed outline follows as Section 1.2), in the rest of the Introduction we introduce the separation results and show how applying them naively, using the separator

alone, gives an algorithm worse than existing ones for MAX 2-CSP; perhaps this is why such algorithms have not already been developed. Then, we sketch how, under optimistic assumptions, using small separators *in conjunction with* existing “Measure and Conquer” analyses gives an improvement. In the body of the paper we turn this into a rigorous analysis of a faster new algorithm for MAX 2-CSP. We then describe a general framework for this “Separate, Measure and Conquer” approach to the design and analysis of polynomial-space, exponential-time algorithms based on small balanced separators, and use it to obtain faster algorithms for #DOMINATING SET, calling upon the method in significantly greater generality than was needed for MAX 2-CSP. We conclude with some thoughts on the method, including the likely limitation that the (polynomial-space) algorithms based on it will be slower than exponential-space dynamic programming algorithms based on small pathwidth.

We now introduce the main separation results we will use. The *bisection width* of a graph is the minimum number of edges between vertices belonging to different parts, over all partitions of the vertex sets into two parts whose size differs by at most one. Monien and Preis [MP01, MP06] proved the following upper bound on the bisection width of cubic graphs.

Theorem 1 ([MP06]). *For any $\epsilon > 0$ there is a value $n(\epsilon)$ such that the bisection width of any cubic graph $G = (V, E)$ with $|V| > n(\epsilon)$ is at most $(\frac{1}{6} + \epsilon)|V|$.*

As noted in [FH06], the bound also holds for graphs of maximum degree at most 3 and a corresponding bisection can be computed in polynomial time. Bezrukov et al. [BEM⁺04] showed a lower bound of $0.082n$ on the maximum bisection width of cubic graphs.

Let (L, S, R) be a partition of the vertex set of a graph $G = (V, E)$ such that there is no edge in G with one endpoint in L and the other endpoint in R . We say that (L, S, R) is a *separation* of G , and that S is a *separator* of G , *separating* L and R (thought of as Left and Right). The previous theorem immediately gives a small “balanced” separator, one partitioning the remaining vertices into equal-sized parts. It is obtained by choosing a vertex cover of the edges in the bisection.

Lemma 2. *For any $\epsilon > 0$ there is a value $n(\epsilon)$ such that every graph $G = (V, E)$ of maximum degree at most 3, $|V| \geq n(\epsilon)$, has a separation (L, S, R) with $|S| \leq (\frac{1}{6} + \epsilon)|V|$ and $|L|, |R| \leq \left\lceil \frac{|V| - |S|}{2} \right\rceil$. Moreover, such a separation can be computed in polynomial time.*

A direct application of branching on the vertices in such a separator yields algorithms inferior to existing ones. We illustrate with solving a cubic MAX 2-CSP instance with n vertices. We separate the vertices as (L, S, R) per Lemma 2. Sequentially, we generate $r^{|S|}$ smaller problem instances by taking each possible assignment to the vertices of S . As for many problems, for MAX 2-CSP, assigning a value to a vertex $v \in V$ yields an instance on vertices $V \setminus \{v\}$, so the procedure above produces instances on the vertex set $L \cup R$. The restriction of G to those vertices, $G|_{L \cup R}$, is by construction a graph with no edge between L and R , and the solution to an instance on constraint graph $G|_{L \cup R}$ is simply the direct combination of the solutions to the corresponding instances given by $G|_L$ and $G|_R$. The advantage given by reducing on vertices in the *separator* is that (for each branch) we must solve two small instances rather than one large one. Roughly speaking, in the worst case where $|S| = \frac{1}{6}n$ and $|L| = |R| = \frac{5}{12}n$, the running time $t(n)$ satisfies the recurrence

$$t(n) = r^{\frac{1}{6}n} (2 \cdot t(\frac{5}{12}n)),$$

whose solution satisfies $t(n) = O^*(r^{\frac{2}{7}n})$.¹ This is inferior to the $O^*(r^{n/4})$ upper bound from [SS06, SS07] for a simple polynomial-space algorithm based on local simplification and branching rules.

¹As is conventional, we use the notation $f(n) = O^*(g(n))$ to indicate that there exists a polynomial $p(n)$ such that for sufficiently large n , $f(n) \leq p(n)g(n)$.

The improvements in this paper have their origin in a simple observation: if a branching algorithm can always “pivot” on vertices in the separator, then the usual measure of improvement is achieved at each step (typically computed by the Measure and Conquer method described in Subsection 2.3), and the splitting of the graph into two parts when the separator is emptied is a bonus. We get the best of both. (The technical challenges are to accurately amortize this bonus over the previous branches to prove a better running time, and to control the balance of the separation as the algorithm proceeds.)

Again, we illustrate for cubic MAX 2-CSP. As remarked earlier, we will be very optimistic in this sketch, doing the analysis rigorously in Section 3. Again, let us branch on (assign a value to) a vertex $v \in S$. It is possible that v has one or more neighbors within S , but this is a favorable case, reducing the number of subsequent branches needed. So, suppose that v has neighbors only in L and R . If all its neighbors were in one part, the separator could be made smaller, so let us skip over this case as well. The cases of interest, then, are when v has two neighbors in L and one in R , or vice-versa. Suppose that these cases occur equally often; this is the bit of optimism that will require more care to get right. In that case, after all $|S|$ branchings, the sizes of L and R are each reduced by $\frac{3}{2}|S|$. This would lead to a running time bound $t(n)$ satisfying the recurrence

$$t(n) = r^{\frac{1}{6}n} \cdot 2t(\frac{5}{12}n - \frac{3}{2} \cdot \frac{1}{6}n),$$

leading to a solution satisfying $t(n) = O^*(r^{\frac{1}{5}n})$. This conjectured bound would improve on the best previous algorithm’s time bound of $O^*(r^{\frac{1}{4}n})$, and Section 3 establishes that the bound is true, modulo a subexponential factor in the running time due to Lemma 2 guaranteeing only $|S| \leq (\frac{1}{6} + \epsilon)n$ instead of $|S| \leq \frac{1}{6}n$.

Our algorithms will exploit a *global* graph structure, the separator, while executing an algorithm based on *local* simplification and branching rules. The use of global structure may also make it possible to circumvent lower bounds existing for certain classes of algorithms that are typically restricted to local information [AS00, AHI05].

1.2. Results and organization of the paper. Section 2 defines notation and gives the necessary background on separators and the Measure and Conquer method that are necessary for Section 3. Section 3 gives a first analysis of a separator-based algorithm. It solves cubic MAX 2-CSP instances in time $r^{(1/5+o(1))n}$, where n is the number of vertices of the input instances. This outperforms the previously fastest $O^*(r^{n/4})$ time algorithm [SS07].² It also speeds up the fastest known algorithm for general MAX 2-CSP instances from $O^*(r^{(19/100)m})$ to $r^{(9/50+o(1))m}$. The same time bounds apply to Polynomial and Ring CSP (encompassing graph bisection, the Ising model, and counting).

Subsection 3.4 gives tight lower bounds for the analysis of the previously fastest MAX 2-CSP algorithm. These lower bounds highlight that our new algorithm is strictly faster. It should be noted that tight analyses are extremely rare for competitive branching algorithms. An important feature of these lower bounds is that they make the algorithm use various branching rules whose Measure and Conquer analyses are most constraining.

While MAX 2-CSP is a central problem in exponential-time algorithmics, the analysis of its branching algorithms is typically easier than for other problems. This is largely due to the branching rule creating isomorphic subinstances. In Section 4, we develop the Separate, Measure and Conquer method in full generality without relying on the simplifying features of MAX 2-CSP. We further illustrate the method in Section 5 where we use it to design faster polynomial space algorithms for counting dominating sets in graphs, both for graphs with maximum degree at most 3 and for

²There is an exponential-space algorithm with a better worst-case running time [SS07] and for vertex-parameterized running times, William’s [Wil05] exponential space algorithm remains unbeaten, but we restrict the discussion to polynomial-space algorithms in this subsection.

general graphs. For cubic graphs, we obtain a time bound of $3^{(1/6+o(1))n} = O(1.2458^n)$, improving on the previous best $O^*(2^{(1/2)n}) = O(1.4143^n)$ [KMRR05]. For general graphs, our new time bound is $O(1.5183^n)$, improving on the previous best $O(1.5673^n)$ time algorithm [vR10].

Limitations of the Separate, Measure and Conquer method are discussed in Section 6, and we provide an outlook in the Conclusion.

The hasty reader may choose to skip Subsections 3.4, 5.1, and/or 5.2, or Section 5 altogether.

2. PRELIMINARIES

2.1. Graphs. Let $G = (V, E)$ be a (simple, undirected) graph, $v \in V$ be a vertex and $S \subseteq V$ a vertex subset of G . We also refer to the vertex set and edge set of G by $V(G)$ and $E(G)$, respectively. We denote the *open* and *closed neighborhoods* of v by $N_G(v) = \{u \in V : \{u, v\} \in E\}$ and $N_G[v] = N_G(v) \cup \{v\}$, respectively. The *closed neighborhood* of S is $N_G[S] = \bigcup_{v \in S} N_G[v]$ and its *open neighborhood* is $N_G(S) = N_G[S] \setminus S$. The set S and vertex v are said to *dominate* the vertices in $N_G[S]$ and $N_G[v]$, respectively. The set S is a *dominating set* of G if $N_G[S] = V$. The degree of v in G is $d_G(v) = |N_G(v)|$. We sometimes omit the subscripts when they are clear from the context. The *contraction* of an edge $uv \in E$ is an operation replacing both vertices u and v by one new vertex c_{uv} that is adjacent to $N_G(\{u, v\})$. The graph $G - S$ is obtained from G by removing the vertices in S and all incident edges. When $S = \{v\}$ we also write $G - v$ instead of $G - \{v\}$. The graph induced on S is $G[S] = G - (V \setminus S)$. The graph G is *cubic* or 3-regular if each vertex has degree 3 and *subcubic* if each vertex has degree at most 3.

A *tree decomposition* of G is a pair $(\{X_i : i \in I\}, T)$ where $X_i \subseteq V$, $i \in I$, and T is a tree with elements of I as nodes such that:

- (1) for each edge $uv \in E$, there is an $i \in I$ such that $\{u, v\} \subseteq X_i$, and
- (2) for each vertex $v \in V$, $T[\{i \in I : v \in X_i\}]$ is a (connected) tree with at least one node.

The *width* of a tree decomposition is $\max_{i \in I} |X_i| - 1$. The *treewidth* [RS86] of G is the minimum width taken over all tree decompositions of G . *Path decompositions* and *pathwidth* are defined similarly, except that T is restricted to be a path.

2.2. Separators. Our algorithms will typically pivot (or branch) on a vertex of the separator S , separating L and R , producing instances with smaller separators. As reductions might render L much smaller than R , we would sometimes like to rebalance L and R without increasing the size of the separator.

Lemma 3. *Let $G = (V, E)$ be a graph of maximum degree at most 3 and let S be a separator of G separating L and R . If there is a vertex $s \in S$ with exactly one neighbor l in L , then there is a separator S' of G separating L' and R' such that $|S'| = |S|$ and $|L'| = |L| - 1$.*

Proof. We use the partition $(L', S', R') = (L \setminus \{l\}, (S \setminus \{s\}) \cup \{l\}, R \cup \{s\})$. □

We say in this case that we *drag* s into R . This transformation, though it does not change the “true” problem instance, does change its presentation and its measure; it is treated like any other transformation in the Measure and Conquer analysis, to which we turn next.

2.3. Measure and Conquer. In this subsection, we recall the basics of the Measure and Conquer method. The method was introduced with that name by [FGK09] but closely parallels Eppstein’s quasi-convex method and Scott and Sorkin’s linear-programming approach [SS07]. We use the method in a form modeled on our development in [GS12], yielding convex mathematical programs.

To track the progress a branching algorithm makes when solving an instance, a Measure and Conquer analysis assigns a potential function to instances, a so-called measure.

Definition 4. *A measure μ for a problem P is a function from the set of all instances for P to the set of non negative reals.*

A Measure and Conquer analysis upper bounds the running time of a branching algorithm by analyzing the size of the search tree. Two values are of interest, the depth of the search tree and its number of leaves. In the following theorem, the measure η is typically polynomially bounded and used to bound the depth, and μ is used to bound the number of leaves.

Lemma 5 ([Gas10, GS12]). *Let A be an algorithm for a problem P , $c \geq 0$ and $r > 1$ be constants, and $\mu(\cdot), \eta(\cdot)$ be measures for the instances of P , such that for any input instance I , A reduces I to instances I_1, \dots, I_k , solves these recursively, and combines their solutions to solve I , using time $O(\eta(I)^c)$ for the reduction and combination steps (but not the recursive solves), with*

$$(\forall i) \quad \eta(I_i) \leq \eta(I) - 1, \text{ and} \quad (1)$$

$$\sum_{i=1}^k r^{\mu(I_i)} \leq r^{\mu(I)}. \quad (2)$$

Then A solves any instance I in time $O(\eta(I)^{c+1})r^{\mu(I)}$.

3. MAX 2-CSP

Using the notation from [SS07], an instance (G, S) of MAX 2-CSP (also called MAX $(r, 2)$ -CSP) is given by a *constraint graph* $G = (V, E)$ and a set S of *score functions*. Writing $[r] = \{1, \dots, r\}$ for the set of available vertex colors, we have a *dyadic* score function $s_e : [r]^2 \rightarrow \mathbb{R}$ for each edge $e \in E$, a *monadic* score function $s_v : [r] \rightarrow \mathbb{R}$ for each vertex $v \in V$, and a single *niladic* score “function” $s_\emptyset : [r]^0 \rightarrow \mathbb{R}$ which takes no arguments and is just a constant convenient for bookkeeping. A *candidate solution* is a function $\phi : V \rightarrow [r]$ assigning colors to the vertices (ϕ is an *assignment* or *coloring*), and its score is

$$s(\phi) := s_\emptyset + \sum_{v \in V} s_v(\phi(v)) + \sum_{uv \in E} s_{uv}(\phi(u), \phi(v)).$$

An *optimal solution* ϕ is one which maximizes $s(\phi)$.

Let us recall the reductions from [SS07]. Reductions 0, I, and II are simplification rules (acting, respectively, on vertices of degree 0, 1, and 2), creating one subinstance, and Reduction III is a branching rule, creating r subinstances. An optimal solution for (G, S) can be found in polynomial time from optimal solutions of the subinstances.

Reduction 0: If $d(y) = 0$, then set $s_\emptyset = s_\emptyset + \max_{C \in [r]} s_y(C)$ and delete y from G .

Reduction I: If $N(y) = \{x\}$, then replace the instance with (G', S') where $G' = (V', E') = G - y$ and S' is the restriction of S to V' and E' except that for all colors $C \in [r]$ we set

$$s'_x(C) = s_x(C) + \max_{D \in [r]} \{s_{xy}(CD) + s_y(D)\}.$$

Reduction II: If $N(y) = \{x, z\}$, then replace the instance with (G', S') where $G' = (V', E') = (V - y, (E \setminus \{xy, yz\}) \cup \{xz\})$ and S' is the restriction of S to V' and E' , except that for $C, D \in [r]$ we set

$$s'_{xz}(CD) = s_{xz}(CD) + \max_{F \in [r]} \{s_{xy}(CF) + s_{yz}(FD) + s_y(F)\}$$

if there was already an edge xz , discarding the first term $s_{xz}(CD)$ if there was not.

Reduction III: Let y be a vertex of degree 3 or higher. There is one subinstance (G', s^C) for each color $C \in [r]$, where $G' = (V', E') = G - y$ and s^C is the restriction of s to V' , except that we set

$$(s^C)_\emptyset = s_\emptyset + s_y(C),$$

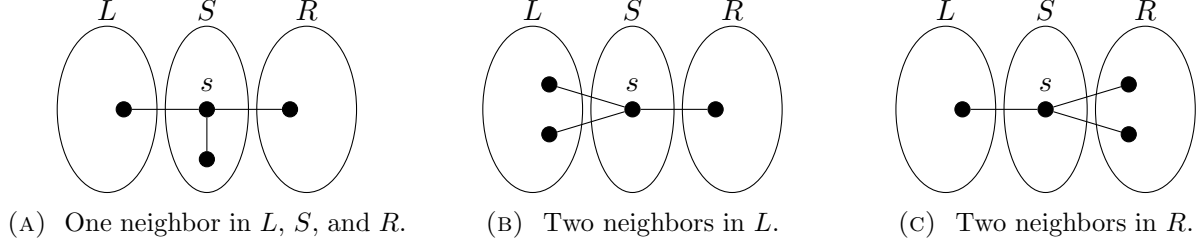


FIGURE 1. Configurations for branching on a separator vertex.

and, for every neighbor x of y and every $D \in [r]$,

$$(s^C)_x(D) = s_x(D) + s_{xy}(DC).$$

We are now going to describe a new separator-based algorithm for cubic MAX 2-CSP, which is faster than the algorithm given by Scott and Sorkin in [SS07]. Using it as a subroutine in the algorithm for general instances by Scott and Sorkin [SS07] also gives a faster running time for MAX 2-CSP on arbitrary graphs.

3.1. Background. For a cubic instance of MAX 2-CSP, an instance whose constraint graph G is 3-regular, the fastest known polynomial-space algorithm makes simple use of the reductions above. The algorithm branches on a vertex v of degree 3, giving two instances with a common constraint graph G' , where v has been deleted from the original constraint graph. In G' , the three G -neighbors of v each have degree 2. Simplification rules are applied to rid G' of degree-2 vertices, and further vertices of degree 0, 1, or 2 that may result, until the constraint graph becomes another cubic graph G'' . This results in two instances with the common constraint graph G'' , to which the same algorithm is applied recursively. The running time of the algorithm is exponential in the number of branchings, and since each branching destroys 4 degree-3 vertices (the pivot vertex v and its three neighbors), the running time is bounded by $O^*(r^{n/4})$; details may be found in [SS06].

For this algorithm, $r^{n/4}$ is also a lower bound, achieved for example by an instance whose constraint graph consists of disjoint copies of K_4 , the complete graph on 4 vertices. In fact, instances with disjoint constraint graphs can be solved with far greater efficiency, since the components can be solved separately, a fact exploited by the fastest polynomial-space algorithms for MAX 2-CSP [SS07]. However, Subsection 3.4 shows that $r^{n/4}$ is also a lower bound for algorithms exploiting connectedness, since a slightly unlucky choice of pivot vertices leaves a particular worst-case constraint graph connected. We conjecture that $r^{n/4}$ is a lower bound for any algorithm using these reductions and choosing its pivot “locally”: that is, characterizing each vertex by the structure of the constraint graph (or even the instance) within a fixed-radius ball around it, and choosing a vertex with a best such character.

Here, we show how to break this $r^{n/4}$ barrier, by selecting pivot vertices using global properties of the constraint graph. In this section we describe an algorithm which pivots only on vertices in a separator S of G . When the separator is exhausted, G has been split into two components L and R which can be solved independently, and are treated recursively. The efficiency gain of the algorithm comes from the component splitting: if the time to solve an instance with n vertices can be bounded by $O^*(r^{cn})$, the time to solve an instance consisting of components L and R is $O^*(r^{c|L|}) + O^*(r^{c|R|})$, which (for L and R of comparable sizes) is hugely less than the time bound $O^*(r^{c(|L|+|R|)})$ for a single component of the same total order. This efficiency gain comes at no cost: until the separator is exhausted, branching on vertices in the separator is just as efficient as branching on any other vertex.

3.2. Analysis. To analyze the algorithm, we use the Measure and Conquer method. As in [GS12], we use penalty terms in the measure to treat tricky cases that otherwise require arguments outside the Measure and Conquer framework. (Those arguments are typically simple, but mesh poorly with the Measure and Conquer framework, making correctness difficult to check. Our penalty approach is modeled on Wahlström’s [Wah04].) We also take from [SS07] and [GS12] the treatment of vertices of degrees 1 and 2 within the Measure and Conquer framework. Reductions on such vertices are *de facto* never an algorithm’s critical cases, but can lead to a tangle of special cases unless treated uniformly.

Recall (see Lemma 5) that the Measure and Conquer analysis applies to an algorithm which polynomially transforms an instance I to one or more instances I_1, \dots, I_k , solves those instances recursively, and obtains a solution to I as a polynomial-computable function of the solutions of I_1, \dots, I_k . The measure $\mu(I)$ of an instance I should satisfy that for any instance,

$$\mu(I) \geq 0, \quad (3)$$

and for any transformation of I into I_1, \dots, I_k ,

$$r^{\mu(I_1)} + \dots + r^{\mu(I_k)} \leq r^{\mu(I)}. \quad (4)$$

Given these hypotheses, the algorithm solves any instance I in time at most $O^*(r^{\mu(I)})$, if the number of recursive calls from the root to a leaf of the search tree is polynomially bounded.

Here, we present an instance of MAX 2-CSP in terms of a separator of its constraint graph $G = (V, E)$: a partition of V into sets S (separator), R (right), and L (left), such that no edge of G has endpoints in both L and R , i.e., $E \cap (L \times R) = \emptyset$. We write R_3 , L_3 , and S_3 for the subset of degree-3 vertices of R , L , and S , respectively, and we will always assume that $|L_3| \leq |R_3|$, if necessary swapping the roles of L and R to make it so. We write $|S_2|$ for the number of degree-2 vertices in S .

We define the measure of an instance as

$$\begin{aligned} \mu(L, S, R) = & w_s |S_3| + w_s^2 |S_2| + w_r |R_3| + \\ & + w_b \mathbb{1}(|R_3| = |L_3|) + w_c \mathbb{1}(|R_3| = |L_3| + 1) + w_d \log_{3/2}(|R_3| + |S_3|), \end{aligned} \quad (5)$$

where the values w_s , w_s^2 , w_r , w_b , w_c , and w_d are constants to be determined and the indicator function $\mathbb{1}(\text{event})$ takes the value 1 if the event is true and 0 otherwise. For the constraint (3) that $\mu \geq 0$, it suffices to constrain each of the constants to be nonnegative:

$$w_s, w_s^2, w_r, w_b, w_c, w_d \geq 0. \quad (6)$$

The term w_b is a penalty applied when the left side is as large as the right: the running-time bound (via μ) only explicitly addresses the right side, and it is natural that a larger bound is needed when the left side’s size is as large as it can be. Similarly, the term w_c penalizes near-equality. (We thus anticipate that $w_b \geq w_c$, as turns out to be true.)

The logarithmic term in the measure accounts for the fact that a new separator may be computed a logarithmic number of times on a path from the root to a leaf in the search tree. Each time a new separator is computed, an instance may go from being very imbalanced to being perfectly balanced and therefore see an increase in penalty terms. The logarithmic term offsets any such penalty increases.

From (4), each reduction imposes a constraint on the measure. Whenever there is a vertex of degree 0, 1, or 2 a corresponding reduction is applied, so it can be presumed that any other reduction is applied to a cubic graph. We treat the reductions in their order of priority: when presenting one reduction, we assume that no previously presented reduction can be applied to the instance.

Reduction 0. If the instance contains a vertex v of degree 0, then perform Reduction 0 on v . Removing v from the instance has no effect on the measure and condition (4) is satisfied.

Half-edge deletion. A half-edge deletion occurs when the degree of a vertex decreases. We will require that a decrease of the degree of a vertex does not increase the measure, which will validate Reduction I, the collapse of parallel edges, and Reduction II for vertices in L and R .

We analyze the effect on the measure when the degree of a vertex v decreases. Denote by μ the value of the measure before the degree decrease and by μ' its value after the degree decrease.

First, assume $d(v) = 1$. The degree of v is reduced to 0. Since neither degree-0 nor degree-1 vertices affect the measure, $\mu' - \mu = 0$, satisfying Condition (4).

Next, assume $d(v) = 2$. If $v \in L$, then $\mu' - \mu = 0$. If $v \in S$, then $\mu' - \mu \leq -w_s^2$, which satisfies Condition (4) since $w_s^2 \geq 0$ by (6). If $v \in R$, then $\mu' - \mu = 0$, which also satisfies Condition (4).

Finally, assume $d(v) = 3$. We consider several cases.

- $v \in L$. Since the imbalance increases by one vertex, $\mu' - \mu \leq \max(0, -w_c, -w_b + w_c)$. Since $w_c \geq 0$ by (6) it suffices to constrain

$$-w_b + w_c \leq 0. \quad (7)$$

- $v \in S$. For this case it is sufficient that

$$-w_s + w_s^2 \leq 0. \quad (8)$$

- $v \in R$. The case where $|R_3| = |L_3|$ is covered by (7) since we can swap L and R . Otherwise, $|R_3| \geq |L_3| + 1$, and then $\mu' - \mu \leq -w_r + \max(0, w_c, w_b - w_c)$. Since $w_c \geq 0$ by (6), it suffices to constrain

$$-w_r + w_c \leq 0 \text{ and} \quad (9)$$

$$-w_r + w_b - w_c \leq 0. \quad (10)$$

Separation. This reduction is the only one special to separation, and its constraint looks quite different from those of previous works. The reduction applies when $S = \emptyset$, which arises in two cases. One is at the beginning of the algorithm, when the instance has not been separated, and may be represented by the trivial separation $(\emptyset, \emptyset, V)$. The second is when reductions on separated instances have exhausted the separator, so that S is empty but both L and R are nonempty, and the instance is solved by solving the instances on L and R independently, via a new separator (L', S', R') for R and another such separator (L'', S'', R'') for L . The reduction is applied to a graph $G = (V, E)$ that is cubic and can be assumed to be of at least some constant order, $|V| \geq k$, since a smaller instance can be solved in constant time. By Lemma 2 we know that, for any constant $\epsilon > 0$, there is a size $k = k(\epsilon)$ such that any cubic graph G of order at least k has a separator (L, S, R) with $|S| \leq (\frac{1}{6} + \epsilon)|V|$, $|S|, |R| \leq \frac{5}{12}|V|$. From (4), making worst-case assumptions about balance, it suffices to constrain that

$$\begin{aligned} & r^{w_s|S'_3|+w_r|R'_3|+w_b+w_d \log(|R'_3|+|S'_3|)} + r^{w_s|S''_3|+w_r|R''_3|+w_b+w_d \log(|R''_3|+|S''_3|)} \\ & \leq r^{w_r|R_3|+w_d \log(|R_3|)}. \end{aligned}$$

From the separator properties, this in turn is implied by

$$2 \cdot r^{w_s(1/6+\epsilon)|R_3|+w_r(5/12|R_3|)+w_b+w_d \log(8/12|R_3|)} \leq r^{w_r|R_3|+w_d \log(|R_3|)},$$

where we have estimated $|L'_3|, |R'_3| \leq \frac{5}{12}|R_3|$ and $|S'_3| \leq (\frac{1}{6} + \epsilon)|R_3| \leq \frac{3}{12}$ in the log term on the left hand side. Since $r \geq 2$, it suffices to constrain that

$$1 + w_s(\frac{1}{6} + \epsilon)|R_3| + w_r(\frac{5}{12}|R_3|) + w_b + w_d \log(\frac{8}{12}|R_3|) \leq w_r|R_3| + w_d \log(|R_3|).$$

Taking $\frac{3}{2} = \frac{12}{8}$ to be the logarithm's base and setting $w_d = w_b + 1$, the left term $w_d \log(\frac{8}{12}|R_3|)$ is equal to $-(w_b + 1) + (w_b + 1) \log(|R_3|)$, and it suffices to have

$$w_s(\frac{1}{6} + \epsilon) + w_r(\frac{5}{12}) \leq w_r. \quad (11)$$

Reduction II in S . If the instance has a vertex $s \in S$ of degree 2, then perform Reduction II on s . Let u_1, u_2 denote the neighbors of s . The vertex s is removed and the edge $u_1 u_2$ is added if it was not present already. Unless $u_1 \in L$ and $u_2 \in R$ (or the symmetric case), where we need to adjust the separator, Condition (4) is implied by the constraints of the half-edge deletions. If $u_1 \in L$ and $u_2 \in R$ (or the symmetric case), then S is not a valid separator any more. The algorithm removes u_2 from R and adds it to S . If $d(u_2) = 2$, we have that $\mu' - \mu \leq 0$. Otherwise, $d(u_2) = 3$ and $\mu' - \mu \leq -w_s^2 + w_s - w_r + \max(0, w_c, w_b - w_c)$. Since $w_c \geq 0$ by (6) it suffices to constrain

$$-w_s^2 + w_s - w_r + w_c \leq 0, \text{ and} \quad (12)$$

$$-w_s^2 + w_s - w_r + w_b - w_c \leq 0. \quad (13)$$

No neighbors in L . If the separator (L, S, R) has a vertex $v \in S$ with no neighbors in L , “drag” v into R , i.e., transform the instance by changing the separator to $(L', S', R') := (L, S \setminus \{v\}, R \cup \{v\})$. It is easily checked that this is a valid separator, with no edge incident on both L and R , and with $|L'_3| \leq |R'_3|$ implied by $|L_3| \leq |R_3|$. Indeed the new instance is no more balanced than the old, so that the difference between the new and old measures is $\mu' - \mu \leq -w_s + w_r$, and to satisfy condition (4) it suffices that

$$-w_s + w_r \leq 0, \quad (14)$$

since an increase in imbalance does not increase the measure by (6) and (7).

No neighbors in R . This case is similar to the previous case, but a vertex $v \in S$ with no neighbor in R is dragged into L .

- If initially we had $|R_3| = |L_3|$, we reverse the roles of L and R and revert to case 3.2.
- If $|R_3| \geq |L_3| + 1$ then the transformation increases $|L_3|$ by 1, decreasing the imbalance by one vertex. Therefore, $\mu' - \mu \leq -w_s + \max(0, w_c, w_b - w_c)$ and we constrain that

$$-w_s + w_c \leq 0, \text{ and} \quad (15)$$

$$-w_s + w_b - w_c \leq 0. \quad (16)$$

With the above cases covered, we may assume that the pivot vertex $s \in S$ has degree 3 and at least one neighbor in each of L and R .

One neighbor in each of L , S , and R . To pivot on a vertex $s \in S$ with one neighbor in each of L , S , and R , do a type-3 reduction on s , deleting it from the constraint graph and thereby reducing the degree of each neighbor to 2.

Since both L and R lose a degree-3 vertex, there is no change in balance and the constraint is

$$1 - 2w_s + w_s^2 - w_r \leq 0. \quad (17)$$

The form and the initial 1 come from the reduction's generating r instances with common measure μ' , so the constraint is $r \cdot r^{\mu'} \leq r^\mu$, or equivalently $1 + \mu' - \mu \leq 0$. The value of $\mu' - \mu$ comes from S losing two degree-3 vertices but gaining a degree-2 vertex, and R losing a degree-3 vertex.

Two neighbors in L . If $s \in S$ has two neighbors in L and one neighbor in R , a type-3 reduction removes s , reduces the degree of a degree-3 vertex in R , and increases the imbalance by one vertex. The algorithm performs a type-3 reduction if $|R_3| \leq |L_3| + 1$, where $\mu' - \mu \leq -w_s - w_r + \max(-w_b + w_c, -w_c)$. Thus, we constrain

$$1 - w_s - w_r - w_b + w_c \leq 0 \text{ and} \quad (18)$$

$$1 - w_s - w_r - w_c \leq 0. \quad (19)$$

If, instead, $|R_3| \geq |L_3| + 2$, then the algorithm drags s into L and its neighbor $r \in R$ into S , replacing (L, S, R) by $(L \cup \{s\}, (S \setminus \{s\}) \cup \{r\}, R \setminus \{r\})$. We need to constrain that $-w_r + \max(w_b, w_c) \leq 0$, which, since $w_c \leq w_b$ by (7), is satisfied if we constrain that

$$-w_r + w_b \leq 0. \quad (20)$$

Two neighbors in R . If $s \in S$ has two neighbors in R and one neighbor in L , the algorithm always performs a type-3 reduction, which removes s , reduces the degree of two degree-3 vertices in R , and decreases the imbalance by one vertex. For the analysis of the case where $|R_3| = |L_3|$, we refer to (18) since L and R are swapped after the reduction. For the other cases, we constrain

$$1 - w_s - 2w_r - w_c + w_b \leq 0 \text{ and} \quad (21)$$

$$1 - w_s - 2w_r + w_c \leq 0. \quad (22)$$

This describes all the constraints on the measure. To minimize the running time proven by the analysis, we minimize w_r , and find the following set of feasible weights:

$$\begin{array}{lll} w_r = 0.2 + \varepsilon & w_s = 0.7 & w_b = 0.2 \\ & w_s^2 = 0.6 & w_c = 0.1 \end{array}$$

All constraints are satisfied and $\mu \leq (0.2 + \varepsilon)n = (1/5 + o(1))n$, by definition of $o(\cdot)$.

It only remains to verify that the depth of the search trees of the algorithm are upper bounded by a polynomial. Since not every reduction removes a vertex (some only modify the separation (L, S, R)), it is crucial to guarantee some kind of progress for each reduction. Since each reduction decreases another measure $\eta(L, S, R, E) := 3|S_3| + 2|R_3| + |L_3| + 2|E|$, by at least one, and this measure is upper bounded by a polynomial, the depth of the search trees is bounded by a polynomial.

3.3. Max 2-CSP result, consequences, and extensions. The MAX 2-CSP algorithm described in [SS07] for solving an arbitrary instance with n vertices and m edges in time $O^*(r^{19m/100})$ relied upon the ability to solve cubic cases in time $O^*(r^{m/6})$. Here we have shown how to solve cubic cases in time $r^{n/5+o(n)}$, or equivalently $r^{2m/15+o(m)}$, and it was observed in [SS07, Theorem 22] that if cubic cases can be solved faster, the running time of the general algorithm is improved correspondingly (as is its application to graphs of maximum degree 4). This leads us to the following theorem.

Theorem 6. *On input of a MAX 2-CSP instance on a constraint graph G with n vertices and m edges, the algorithm described solves G in time $r^{n/5+o(n)} = r^{2m/15+o(m)}$ if G is cubic, time $r^{7m/40+o(m)}$ if G has maximum degree 4, and time $r^{9m/50+o(m)}$ in general, in all cases using only polynomial space.*

This improves respectively on the previous best running times of $O^*(r^{m/6})$, $O^*(r^{3m/16})$, and $O^*(r^{19m/100})$, all from [SS07].

It also improves on the fastest known polynomial-space running times for Max Cut on cubic, maximum degree 4, and general graphs. The Max Cut problem, a special case of MAX 2-CSP, is

to find a bipartition of the vertices of a given graph maximizing the number of edges crossing the bipartition.

Theorem 6 also extends instantly to Polynomial CSP (PCSP) and Ring CSP (RCSP). The terminology is defined in [SS09], and the extensions follow immediately from the fact that the algorithm here depends only on the Reductions 0, I, II, and III. (Other transformations affect the separation and the analysis of the algorithm, but do not change the instance itself.) Consequences include the ability to efficiently solve graph bisection, to count Max r -SAT solutions, find judicious partitions, and compute the Ising partition function; the formulation of these and other problems as PCSPs is given in [SS09]. Specifically, we have the following two theorems.

Theorem 7. *Let R be a ring and let G be a cubic graph with n vertices. Let I be any RCSP over R , with constraint graph G and domain $[k]$. Then the ring extension of the algorithm described here calculates the partition function Z_I in polynomial space and with $k^{n/5+o(n)}$ ring operations.*

Theorem 8. *The PCSP extension of our algorithm solves any polynomially bounded PCSP instance (or finds the pruned partition function of any prunable PCSP instance) with m clauses, and over domain $[k]$, in time $k^{9m/50+o(m)}$ and space $O^*(1)$.*

Several authors have designed algorithms for MAX 2-CSP analyzed with respect to the number of vertices and the average degree d . The currently fastest among them is [GK14] with a running time of $O^*\left(r^{n \cdot (1 - \frac{3}{d+1})}\right)$. We improve on this running time by using our algorithm from Theorem 6 to solve degree 5 instances.

Theorem 9. *There is an algorithm, which, on input of a MAX 2-CSP instance on a constraint graph G with n vertices and average degree $d \geq 5$, solves G in time $r^{n \cdot (1 - \frac{3}{d+1}) + o(n)}$ using only polynomial space.*

Proof sketch. It suffices to follow the proof of [GK14, Theorem 2] using the running time of Theorem 6 for degree 5 instances. \square

3.4. Lower Bounds for the Scott-Sorkin Algorithm. Scott and Sorkin [SS07] analyzed the running time of their MAX 2-CSP algorithm with respect to m , the number of edges of the input instance, and showed the following upper bounds:

- $O^*(r^{m/6})$ for instances with maximum degree three,
- $O^*(r^{3m/16})$ for instances with maximum degree four, and
- $O^*(r^{19m/100})$ for instances with no degree restrictions.

In this subsection we prove matching lower bounds. Define $f(n) = \Theta^*(g(n))$ if $f(n) = O^*(g(n))$ and $f(n) = \Omega(g(n))$. By Lemmas 11–13, proved hereafter, and the analysis from [SS07], we obtain the following theorem.

Theorem 10. *The worst-case running time of the Scott-Sorkin algorithm for MAX 2-CSP is*

- $\Theta^*(r^{m/6})$ for instances with maximum degree 3,
- $\Theta^*(r^{3m/16})$ for instances with maximum degree 4, and
- $\Theta^*(r^{19m/100})$ for instances with no degree restrictions,

where m is the number of edges in the input instance, even when the input instance is connected.

The lower bounds are established by Lemmas 11–13, starting with instances with maximum degree 3. These lemmas define infinite families of instances with maximum degrees 3, 4, and 5, and prove that the algorithm may perform $\Omega(r^{m/6})$, $\Omega(r^{3m/16})$, and $\Omega(r^{19m/100})$ steps, respectively. From the dual solution in the LP analysis in [SS07], it follows that the $r^{19m/100}$ time bound for general instances rests on the algorithm branching on vertices of degrees 5, 4, and 3 in the proportion

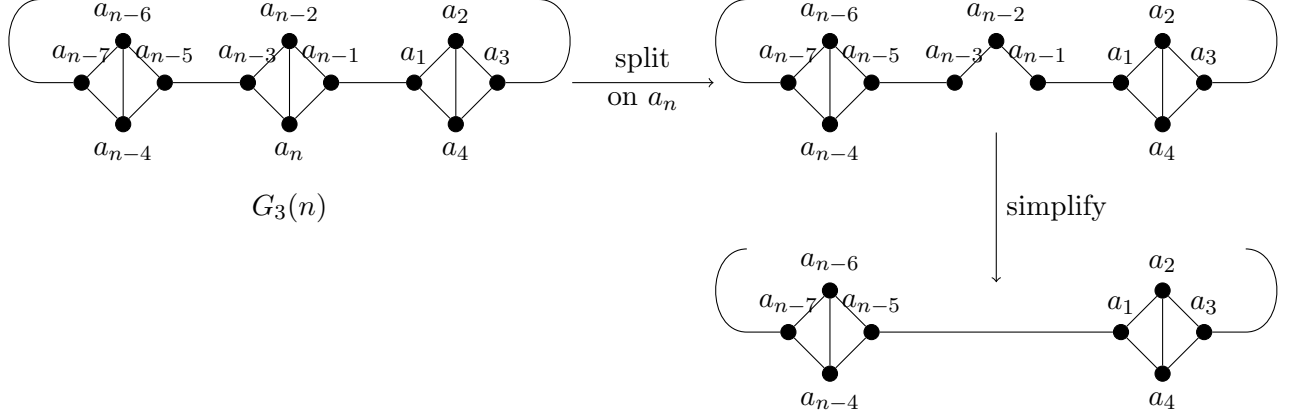


FIGURE 2. The constraint graph $G_3(n)$ and how it evolves when branching on vertex a_n .

8 to 6 to 5, and that no additional Reductions 0, I, and II occur beyond the ones directly needed by Reduction III. The LP analysis does not indicate how to go about constructing instances for which this occurs, nor proves their existence. The running time lower bound of Lemma 13 is proved by explicitly constructing such instances.

Lemma 11. *On connected instances with maximum degree 3, the Scott-Sorkin algorithm has worst-case running time $\Omega(r^{m/6})$.*

Proof. The lemma is proven by exhibiting an infinite family of connected 3-regular instances such that the algorithm may perform $r^{n/4}$ steps when given an instance on n vertices from this family.

Let n be an integer that is divisible by 4. Consider the execution of the Scott-Sorkin algorithm on a MAX 2-CSP instance whose constraint graph is the graph $G_3(n)$ from Figure 2. It is obtained from a cycle $(a_1, a_2, a_3, a_5, a_6, a_7, \dots, a_{n-3}, a_{n-2}, a_{n-1})$ on $3n/4$ vertices by adding $n/4$ new vertices a_4, a_8, \dots, a_n and the edges $a_{4i}a_{4i-3}, a_{4i}a_{4i-2}, a_{4i}a_{4i-1}, 1 \leq i \leq n/4$.

The algorithm selects an arbitrary vertex of degree 3 and splits on it (i.e., it performs Reduction III on it). Suppose the algorithm selects a_n .

Let us first show that the constraint graph that is obtained by performing Reduction III on a_n and simplifying the instance is $G_3(n-4)$. If $n = 4$, observe that $G_3(4)$ is a complete graph on 4 vertices. Branching on one vertex leaves a complete graph on 3 vertices which vanishes by applications of Reduction II, Reduction I and Reduction 0. If $n > 4$, the algorithm splits on variable a_n , which creates two instances where a_n is removed. Afterwards, Reduction II is performed on the variables a_{n-3}, a_{n-2} , and a_{n-1} , and the constraint graph of the resulting instances is $G_3(n-4)$.

Thus, Reduction III is executed $n/4 = m/6$ times recursively by the algorithm, resulting in a running time of $\Omega(r^{m/6})$. \square

Lemma 12. *On connected instances with maximum degree 4, the Scott-Sorkin algorithm has worst-case running time $\Omega(r^{3m/16})$.*

Proof. The lemma is proven by exhibiting an infinite family of connected instances with maximum degree 4, where only a constant number of vertices have degree less than 4, such that the algorithm may perform $\Omega(r^{3n/8})$ steps when given an instance on n vertices from this family.

The graph family will use the following construction. The graph $G_4(n_3, n_4)$, with n_3 divisible by 4 and $n_4 \leq n_3$ divisible by 2, is obtained from $G_3(n_3)$ by the following modifications. Let $M = \{a_1a_2, a_3a_4, \dots, a_{n_4-1}a_{n_4}\}$ and observe that M is a matching in $G_3(n_3)$. Remove the edges in M from the graph. Add a path $(x_2, x_3, \dots, x_{n_4})$ on $n_4 - 1$ new vertices to the graph, and add

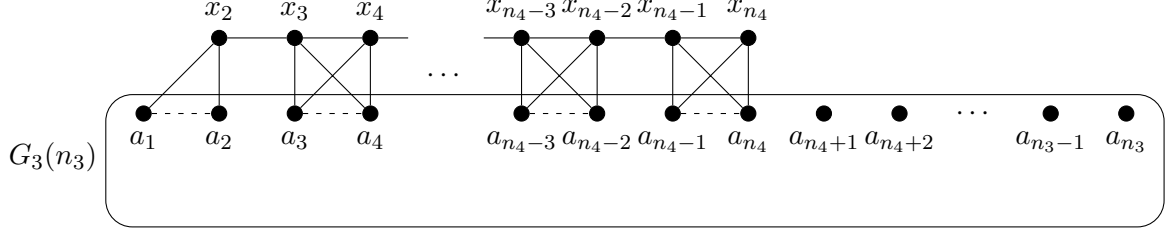


FIGURE 3. The graph $G_4(n_3, n_4)$, with n_3 divisible by 4 and $n_4 \leq n_3$ divisible by 2, is obtained from $G_3(n_3)$ by removing the matching $\{a_1a_2, \dots, a_{n_4-1}a_{n_4}\}$ (dashed edges), and adding $n_4 - 1$ new vertices x_2, x_3, \dots, x_{n_4} and the depicted edges.

the edges $x_i a_i$, $2 \leq i \leq n_4$, the edge $x_2 a_1$, and the edges $x_i a_{i-1}$, $x_{i-1} a_i$, for all even $i = 4, 6, \dots, n_4$. See Figure 3.

On graphs with maximum degree 4, the Scott-Sorkin algorithm performs Reduction III on vertices of degree 4, with a preference for those vertices of degree 4 that have neighbors of degree 3. Assume that $n_4 \geq 4$ and that the algorithm splits on x_{n_4-1} . We claim that this creates two instances, both with constraint graph $G_4(n_3, n_4 - 2)$. Indeed, branching on x_{n_4-1} creates a constraint graph where x_{n_4-1} is removed, which triggers Reduction II on x_{n_4} , resulting in $G_4(n_3, n_4 - 2)$. If $n_4 = 2$, then Reduction II applies to x_2 , creating $G_4(n_3, 0) = G_3(n_3)$.

We conclude that Reduction III is executed $n_4/2 - 2$ times recursively by the algorithm before reaching instances with constraint graph $G_3(n_3)$, for which the running time is characterized by Lemma 11. The algorithm may therefore execute

$$r^{n_4/2-2} \cdot r^{n_3/4} \quad (23)$$

steps. Setting $n_3 = n_4$, we obtain a running time of $\Omega(r^{(n/2-1)/2-2} \cdot r^{n/8}) = \Omega(r^{3n/8}) = \Omega(r^{3m/16})$. \square

Lemma 13. *On connected instances with maximum degree 5, the Scott-Sorkin algorithm has worst-case running time $\Omega(r^{19m/100})$.*

Proof. The lemma is proven by exhibiting an infinite family of connected instances with maximum degree 5, where only a constant number of vertices have degree less than 5, such that the algorithm may perform $r^{19n/40}$ steps when given an instance on n vertices from this family.

Let n be an integer that is divisible by 40. Let $n_3 = n/5$, $n_4 = 3n/5$, and $n_5 = n/5$. To construct the constraint graph $G_5(n)$, we start with $G_4(n_3 + n_4/2, n_4/2)$ (the construction is given in the proof of Lemma 12). Add n_5 new vertices y_1, y_2, \dots, y_{n_5} and edges to form a cycle $(y_1, a_{n_4/2+1}, y_2, a_{n_4/2+2}, \dots, y_{n_5}, a_{n_4/2+n_3})$ (observe that $a_{n_4/2+1}, \dots, a_{n_4/2+n_3}$ all had degree 3 before adding this cycle). Then, for each vertex y_i , $1 \leq i \leq n_5$, add three incident edges, connecting y_i to 3 vertices from $\{a_1, \dots, a_{n_4}\} \cup \{x_2, \dots, x_{n_4}\}$ in such a way that no vertex has degree greater than 5 and y_1 is incident to a vertex of degree 4. See Figure 4.

On graphs with maximum degree 5, the Scott-Sorkin algorithm performs Reduction III on vertices of degree 5, with a preference for those vertices of degree 5 that have neighbors of degree 3 or 4. When there are several choices, we always assume that the algorithm splits on a y vertex with minimum index. Observe that the algorithm splits on y_1, y_2, \dots, y_{n_5} , which leaves the graph $G_4(n_3 + n_4/2, n_4/2)$. Thus, by (23), the overall running time is

$$\Omega(r^{n_5} \cdot r^{(n_4/2)/2-2} \cdot r^{(n_3+n_4/2)/4}) = \Omega(r^{n/5+3n/20-2+n/20+3n/40}) = \Omega(r^{19n/40}) = \Omega(r^{19m/100}) .$$

\square

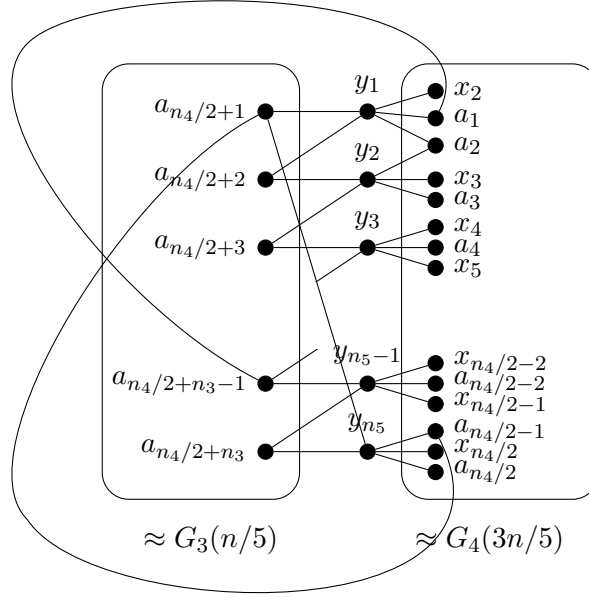


FIGURE 4. The graph $G_5(n)$ is obtained from $G_4(n/2, 3n/10)$ by adding an independent set $\{y_1, \dots, y_{n/5}\}$ and the depicted edges.

It should be noted that tight running time bounds are extremely rare for competitive branching algorithms. Typically, lower bounds proved for branching algorithms are much simpler, using only one branching rule and making sure that this branching rule is applied until the instance has constant size. In the lower bound of this section, however, we managed to make the algorithm branch according to the mixture of branching rules leading to the upper bound — the mixture given by the LP's dual solution. Our message here is twofold. First, when designing lower bounds, it is an advantage to exploit several worst-case branchings of a Measure and Conquer analysis. And second, it could well be that many more existing Measure and Conquer analyses are tight.

4. THE SEPARATE, MEASURE AND CONQUER TECHNIQUE

Our MAX 2-CSP algorithm illustrates that one can exploit separator-based branching to construct a more efficient exponential-time algorithm. However, the MAX 2-CSP problem and its algorithms have certain features that make the analysis simpler than for others. First, Reduction III produces r instances with exactly the same constraint graph, and therefore the same measure. Typically, the instances produced by a reduction rule have different measures. Second, the measure of R depends only on the number of degree-3 vertices in R . This implies a discretized change in the measure for R whenever L and R are swapped. If the measure attaches incommensurable weights to vertices of different types (different degrees, perhaps), then the change in measure resulting from swapping L and R could take values dense within a continuous domain. Our simple measure for the MAX 2-CSP algorithm also implies that the initial separator only need to balance the *number* of vertices in L and R to balance the *measure* of L and R , which is what is needed more generally. Finally, a general method is needed to combine the separator-based branching, which would typically be done for instances with small maximum degree, with the general case, where degrees are not restricted.

In this section we propose a general method to exploit a separator-based branching in the analysis of an algorithm. It will take into account all the complications mentioned, and we will illustrate its use in the next section to analyze a faster algorithm for counting dominating sets. The technique will apply to recursive algorithms that label vertices of a graph, and where an instance can be decomposed into two independent subinstances when all the vertices of a separator have been labeled in a certain way.

Let $G = (V, E)$ be a graph and $\ell : V \rightarrow L$ be a labeling of its vertices by labels in the finite set L . (Partial labelings are handled by including a label whose interpretation is “unlabeled”.) For a subset of vertices $W \subseteq V$, denote by $\mu_r(W)$ and $\mu_s(W)$ two measures for the vertices in W in the graph G labeled by ℓ . The measure μ_r will be used for the vertices on the right hand side of the separator and μ_s for the vertices in the separator. Let (L, S, R) be a partition of V such that no edge of G has one endpoint in L and the other endpoint in R . Initially, we use the separation $(L, S, R) = (\emptyset, \emptyset, V)$. We define the measure

$$\mu(L, S, R) = \mu_s(S) + \mu_r(R) + \max\left(0, B - \frac{\mu_r(R) - \mu_r(L)}{2}\right) + (1 + B) \cdot \log_{1+\epsilon}(\mu_r(R) + \mu_s(S)), \quad (24)$$

where $\epsilon > 0$ is a constant greater than 0 that will be chosen small enough to satisfy constraint (27) below, and B is an arbitrary constant greater than the maximum change (increase or decrease) in imbalance in each reduction in the analysis. The *imbalance* of an instance is $\mu_r(R) - \mu_r(L)$, and we assume, as previously, that

$$\mu_r(R) \geq \mu_r(L). \quad (25)$$

It is important that B be an absolute constant: although the imbalance of an instance may change arbitrarily in an execution of an algorithm, our value of B is only constrained to be greater than the maximum change in imbalance that the analysis takes into account.³ We need two more assumptions about the measure so that it will be possible to compute a balanced separator. Namely, we will assume that adding a vertex to R (and by symmetry, removing a vertex from R) changes the measure $\mu_r(R)$ by at most a constant. The value of this constant is not crucial for the analysis; so we assume for simplicity that it is at most B (adjusting the value of B if necessary):

$$|\mu_r(R \cup \{v\}) - \mu_r(R)| \leq B \quad \text{for each } R \subseteq V \text{ and } v \in V. \quad (26)$$

Moreover, we assume that $\mu_r(R)$ can be computed in time polynomial in $|V|$ for each $R \subseteq V$.

Let us now look more closely at the measure (24). The terms $\mu_s(S)$ and $\mu_r(R)$ naturally define measures for the vertices in S and R , respectively. No term of the measure directly accounts for the vertices in L ; we merely enforce that $\mu_r(R) \geq \mu_r(L)$. The term $\max\left(0, B - \frac{\mu_r(R) - \mu_r(L)}{2}\right)$ is a penalty term based on how balanced the instance is: the more balanced the instance, the larger the penalty term. The penalty term has become continuous, varying from 0 to B . The exact definition of the penalty term will become clearer when we discuss the change in measure when branching. The final term, $(1 + B) \cdot \log_{1+\epsilon}(\mu_r(R) + \mu_s(S))$, amortizes the increase in measure of at most B due to the balance terms each time the instance is separated.

Let us now formulate some generic constraints that the measure should obey. The first one concerns the separation reduction.

4.1. Separation. We assume that an instance with a separation (L, S, R) can be separated into two independent subinstances (L, S, \emptyset) and (\emptyset, S, R) when the labeling of S is contrived enough for separation. Assume this separation can be done when all vertices in S have been labeled

³Typically, if the imbalance decreases by a very large number for a given branching, it is sufficient to replace this number by a large absolute constant without compromising the quality of the analysis.

by a subset $L_s \subseteq L$ of so-called *separation* labels. These labels are very algorithm-specific. For example, a Dominating Set algorithm might have $L_s = \{i, o_d, o_R, o_L\}$, separating the instance when all vertices in the separator have been restricted to be either in the dominating set (i), not in the dominating set and already dominated (o_d), not in the dominating set and needing to be dominated by a vertex in R (o_R), or not in the dominating set and needing to be dominated by a vertex in L (o_L). The separation reduction applies when $\ell(s) \in L_s$ for each $s \in S$, which arises in two cases. The first is at the beginning of the algorithm when the graph has not been separated, which is represented by the trivial separation $(\emptyset, \emptyset, V)$. The second is when our reductions have produced a separable instance, typically with L and R (and possibly S) nonempty.

Let (L, S, R) be such that $\ell(s) \in L_s$ for each $s \in S$. The algorithm separates the instance into the two independent subinstances (L, S, \emptyset) and (\emptyset, S, R) . The algorithm solves these subinstances recursively. Let us focus on the instance (\emptyset, S, R) ; the treatment of the other instance is symmetric. After a cleanup phase, where the algorithm applies some simplification rules (for example, the vertices labelled o_L and needing to be dominated by a vertex in L can be removed in this subinstance), the next step is to compute a new separator of $S \cup R$. This can be done in various ways, depending on the graph class. Let us assume that we are dealing with graphs of bounded maximum degree.

Lemma 14. *Let $G = (V, E)$ be a graph of maximum degree at most Δ , $3 \leq \Delta \leq 6$ and μ be a measure as in (24) satisfying (26) such that $\mu_r(R)$ can be computed in time polynomial in $|V|$ for each $R \subseteq V$. A separation (L, S, R) of G can be computed in polynomial time such that $|\mu_r(L) - \mu_r(R)| \leq B$ and $|S| \leq \alpha_\Delta |V| + o(|V|)$, where $\alpha_3 = 1/6$, $\alpha_4 = 1/3$, $\alpha_5 = 13/30$, and $\alpha_6 = 23/45$.*

Proof. By [FGSS09], the pathwidth of G is at most $\alpha_\Delta |V| + o(|V|)$ and a path decomposition of that width can be computed in polynomial time. We view a path decomposition as a sequence of bags (B_1, \dots, B_b) which are subsets of vertices such that for each edge of G , there is a bag containing both endpoints, and for each vertex of G , the bags containing this vertex form a non-empty consecutive subsequence. The width of a path decomposition is the maximum bag size minus one. We may assume that every two consecutive bags B_i, B_{i+1} differ by exactly one vertex, otherwise we insert between B_i and B_{i+1} a sequence of bags where the vertices from $B_i \setminus B_{i+1}$ are removed one by one followed by a sequence of bags where the vertices of $B_{i+1} \setminus B_i$ are added one by one; this is the standard way to transform a path decomposition into a *nice* path decomposition of the same width where the number of bags is polynomial in the number of vertices [BK96]. Note that each bag is a separator and a bag B_i defines the separation $(L_i, B_i, (\bigcup_{j=i+1}^b B_j) \setminus B_i)$ with $L_i = (\bigcup_{j=1}^{i-1} B_j) \setminus B_i$ and $R_i = V \setminus (L_i \cup B_i)$. Since the first of these separations has $L_1 = \emptyset$ and the last one has $R_b = \emptyset$, at least one of these separations has $|\mu_r(L_i) - \mu_r(R_i)| \leq B$ by (26). Finding such a bag can clearly be done in polynomial time. \square

After a balanced separator (L', S', R') has been computed for $S \cup R$, the instance is solved recursively, and so is the instance $L \cup S$, separated into (L'', S'', R'') . Both solutions are then combined into a solution for the instance $L \cup S \cup R$. Without loss of generality, assume $\mu(L', S', R') \geq \mu(L'', S'', R'')$. Assuming that the separation and combination are done in polynomial time, the imposed constraint on the measure is

$$2 \cdot 2^{\mu_r(R') + \mu_s(S') + B + (1+B) \cdot \log_{1+\epsilon}(\mu_r(R') + \mu_s(S'))} \leq 2^{\mu_r(R) + \mu_s(S) + (1+B) \cdot \log_{1+\epsilon}(\mu_r(R) + \mu_s(S))}$$

To satisfy the constraint, it suffices to satisfy the two constraints:

$$\begin{aligned} \mu_r(R') + \mu_s(S') &\leq \mu_r(R) + \mu_s(S) \\ (1+B) + (1+B) \cdot \log_{1+\epsilon}(\mu_r(R') + \mu_s(S')) &\leq (1+B) \cdot \log_{1+\epsilon}(\mu_r(R) + \mu_s(S)) \end{aligned}$$

To satisfy both constraints, it is sufficient to constrain that

$$\mu_r(R) + \mu_s(S) \geq (1 + \epsilon)(\mu_r(R') + \mu_s(S')). \quad (27)$$

This is the only constraint involving the size of a separation. It constrains that separating (\emptyset, S, R) to (L', S', R') should reduce $\mu_r(R) + \mu_s(S)$ by a constant factor, namely $1 + \epsilon$.

4.2. Branching. The branching rules will apply to two kinds of instances: balanced and imbalanced instances. We say that an instance is *balanced* if $0 \leq \mu_r(R) - \mu_r(L) \leq 2B$ and *imbalanced* if $\mu_r(R) - \mu_r(L) > 2B$. For a balanced instance, the measure (24) is

$$\mu(L, S, R) = \mu_s(S) + \frac{\mu_r(R)}{2} + \frac{\mu_r(L)}{2} + B + (1 + B) \cdot \log_{1+\epsilon}(\mu_r(R) + \mu_s(S))$$

and for an imbalanced instance, it is

$$\mu(L, S, R) = \mu_s(S) + \mu_r(R) + (1 + B) \cdot \log_{1+\epsilon}(\mu_r(R) + \mu_s(S)).$$

Suppose a branch taking (L, S, R, ℓ) to (L', S', R', ℓ') decreases $\mu_r(R) + \mu_r(L)$ by d . Since the measure includes roughly (and at least) half of $\mu_r(R) + \mu_r(L)$, ideally $\mu_r(R) + \max\left(0, B - \frac{\mu_r(R) - \mu_r(L)}{2}\right)$ will decrease by $d/2$. We will now show that this is indeed the case for our measure if the branching guarantees the following balance condition:

$$\text{If } \mu_r(R) - \mu_r(L) > B, \text{ then } \mu_r(R) - \mu_r(R') \geq \mu_r(L) - \mu_r(L'). \quad (28)$$

Recall that B was defined to be greater than the change in imbalance in the analysis of each reduction, so $\mu_r(R) - \mu_r(L) \leq B$ implies the instance remains balanced after a reduction. Condition (28) is very natural, expressing that, if the instance is imbalanced or risks to become imbalanced we would like to make more progress on the large side.

In the balanced case, $0 \leq \mu_r(R) - \mu_r(L) \leq 2B$. If $\mu_r(R) - \mu_r(L) > B$, then by (28), $\mu_r(R') - \mu_r(L') \leq \mu_r(R) - \mu_r(L) \leq 2B$, while if $\mu_r(R) - \mu_r(L) \leq B$, then by the definition of B , $\mu_r(R') - \mu_r(L') \leq 2B$. It follows that the resulting instance (L', S', R', ℓ') is balanced as well, and $\mu_r(R) + \max\left(0, B - \frac{\mu_r(R) - \mu_r(L)}{2}\right)$ decreases by $(\mu_r(R) + \mu_r(L) - \mu_r(R') - \mu_r(L'))/2 = d/2$.

In the imbalanced case, $\mu_r(R) - \mu_r(L) > 2B$. By condition (28), $\mu_r(R) + \max\left(0, B - \frac{\mu_r(R) - \mu_r(L)}{2}\right)$ decreases by

$$\begin{aligned} \mu_r(R) - \mu_r(R') &\geq (\mu_r(R) - \mu_r(R') + \mu_r(L) - \mu_r(L'))/2 \\ &= d/2 \end{aligned}$$

if (L', S', R', ℓ') is also imbalanced, or by

$$\begin{aligned} \mu_r(R) - \left(\frac{\mu_r(R')}{2} + \frac{\mu_r(L')}{2} + B\right) &= \frac{\mu_r(R)}{2} - \left(B - \frac{\mu_r(R)}{2}\right) - \frac{\mu_r(R')}{2} - \frac{\mu_r(L')}{2} \\ &\geq (\mu_r(R) + \mu_r(L) - \mu_r(R') - \mu_r(L'))/2 \\ &= d/2 \end{aligned}$$

if (L', S', R', ℓ') is balanced.

Thus, if Condition 28 holds, i.e., we can guarantee more progress on the large side if the instance is imbalanced or risks to become imbalanced, then the analysis is at least as good as a non-separator based analysis, but with the additional improvement due to the separator branching.

4.3. Integration into a standard Measure and Conquer analysis. The Separate, Measure and Conquer analysis will typically be used when the maximum degree of the instance has become sufficiently small that one can guarantee that a small separator exists. We can view the part of the algorithm analyzed with the Separate, Measure and Conquer analysis as a subroutine and integrate it into any other Measure and Conquer analysis. We only need to guarantee that the measure of an instance does not increase when transitioning to the subroutine. Formally, this is done with the following lemma.

Lemma 15 ([Gas10]). *Let A be an algorithm for a problem P , B be an algorithm for (special instances of) P , $c \geq 0$ and $r > 1$ be constants, and $\mu(\cdot), \mu'(\cdot), \eta(\cdot)$ be measures for the instances of P , such that for any input instance I , $\mu'(I) \leq \mu(I)$ and for any input instance I , A either solves P on I by invoking B with running time $O(\eta(I)^{c+1})r^{\mu'(I)}$, or reduces I to instances I_1, \dots, I_k , solves these recursively, and combines their solutions to solve I , using time $O(\eta(I)^c)$ for the reduction and combination steps (but not the recursive solves),*

$$(\forall i) \quad \eta(I_i) \leq \eta(I) - 1, \text{ and} \tag{29}$$

$$\sum_{i=1}^k r^{\mu(I_i)} \leq r^{\mu(I)}. \tag{30}$$

Then A solves any instance I in time $O(\eta(I)^{c+1})r^{\mu(I)}$.

This completes the description of the Separate, Measure and Conquer method. We now illustrate its use to obtain improved algorithms for $\#$ DOMINATING SET.

5. COUNTING DOMINATING SETS

DOMINATING SET and its variants are of central importance in exponential-time algorithms. The DOMINATING SET problem is to compute a dominating set of minimum size of a given graph $G = (V, E)$. The $\#$ DOMINATING SET problem is to compute a function $d : \{0, \dots, |V|\} \rightarrow \mathbb{N}$ such that for each $k \in \{0, \dots, n\}$, $d(k)$ is the number of dominating sets of G of size k . We denote the number of vertices of the input graph by n .

The current fastest polynomial-space algorithm for $\#$ DOMINATING SET is a $O(1.5673^n)$ time algorithm by van Rooij [vR10]. Like many DOMINATING SET algorithms, it transforms the instance into an instance for SET COVER. Given a multiset \mathcal{S} of subsets of a universe \mathcal{U} , a collection of subsets $C \subseteq \mathcal{S}$ is a *set cover* if $\bigcup_{c \in C} c = \mathcal{U}$. The transformation from DOMINATING SET to SET COVER creates an element for each vertex and a subset for each closed neighborhood. While the fastest known algorithms for DOMINATING SET and variants rely on this transformation to SET COVER, the current fastest polynomial-space algorithm for subcubic graphs is a simple $O^*(2^{n/2})$ time algorithm [KMRR05] that works directly on the input graph. (It outperforms the fastest known algorithms on general graphs [FGK09, Iwa12, vRB11] when their analysis is restricted to the subcubic case.)

In this section, we apply the Separate, Measure and Conquer method to design and analyze faster algorithms for $\#$ DOMINATING SET for both subcubic graphs and general graphs. We improve, separately, on both results. For subcubic graphs we design an algorithm working directly on the input graph, where we can essentially reuse the Max (3, 2)-CSP analysis of Section 3. For general graphs, our algorithm is based on the SET COVER translation and essentially just adds separation to [vR10].

5.1. Subcubic graphs. We assume the input graph G has maximum degree at most 3, and we denote by $\Gamma(G)$ the *cubic structure* of G , the unique 3-regular graph obtained from G by exhaustively contracting edges incident to vertices of degree at most 2 and removing isolated vertices.

To solve the $\#$ DOMINATING SET problem, the algorithm applies labels U , N , and C to vertices. We denote by $\ell(v) \in \{U, N, C\}$ the label of vertex v . The algorithm counts the number of sets D for each size k , $0 \leq k \leq |V|$, with the following restrictions according to the label of each vertex V :

- “unlabeled” U : v needs to be dominated by D ;
- “not in” N : v is not in D , but needs to be dominated by D ;
- “covered” C : there is no restriction on v , i.e., v may or may not be in D and v does not need to be dominated by D .

Initially, all vertices are labeled U . There is no label for vertices that do not need to be dominated and do not belong to a dominating set, since they are simply deleted from the graph. Vertices that are added to the dominating set are also removed from the graph and their neighbors are updated to reflect that they do not need to be dominated any more: neighbors labeled U or C get label C , and neighbors labeled N are deleted. When we speak of a dominating set of a labeled graph G , we mean a vertex set containing vertices labeled U and C that dominates all vertices labeled U and N .

The algorithm will satisfy the invariant that degree-3 vertices in G are labeled U . It will compute an $(n + 1)$ -size vector $\#ds_G$ such that the labeled graph G has $\#ds_G[i]$ dominating sets of size i , $0 \leq i \leq n$.

If $|V(\Gamma(G))| \leq A$ for an arbitrary constant $A \geq 2$, the instance is solved in polynomial time as follows. From a trivial tree decomposition of $\Gamma(G)$ with only one bag, obtain a tree decomposition of width at most A for G in polynomial time [Bod98]. Then, use a tree decomposition based algorithm, such as [vRBR09], to solve the instance in polynomial time. The latter treats unlabeled graphs but needs only minor adaptations for labeled graphs.

Analogous to the algorithm in Section 3, we now define a Reduction III. (Analog of Reductions 0, I, and II are not needed here.)

Reduction III: Let x be a vertex of degree 3. Thus, $\ell(x) = U$. There are three subinstances, G_{in} , G_{opt} , and G_{forb} (thought of as “in”, “optional”, and “forbidden”). They are obtained from G as follows.

- In G_{in} , we add x to the dominating set. The vertex x is deleted from the graph, its neighbors labeled U are relabeled C , and its neighbors labeled N are deleted.
- In G_{opt} , we prevent x from being in the dominating sets and we remove the requirement that it needs to be dominated. The vertex x is removed from the graph.
- In G_{forb} , we prevent $N_G[x]$ from being in the dominating sets, i.e., we forbid that x is dominated. Delete the vertices in $N_G(x)$ labeled C , assign label N to the remaining vertices of $N_G(x)$, and delete x .

Observe that the number of dominating sets of size i of G is

$$\#ds(G)[i] = \#ds(G_{\text{in}})[i - 1] + \#ds(G_{\text{opt}})[i] - \#ds(G_{\text{forb}})[i].$$

The algorithm returns the vector $\#ds$.

We observe that this reduction has at least the same effect on $\Gamma(G)$ as the MAX 2-CSP algorithm of Section 3 has on the constraint graph: the vertex x is deleted, and all vertices with degree at most 2 in $\Gamma(G - x)$ are contracted or, even better, deleted. Thus, the algorithm will select the pivot vertex for branching in exactly the same way as the algorithm in Section 3. Since degree-3 vertices are handled by a 3-way branching, the running time analysis for Max (3, 2)-CSP applies for $\#$ DOMINATING SET as well, giving a running time of $3^{n/5+o(n)} = O(1.2458^n)$ for subcubic graphs. This improves on the previously fastest polynomial-space algorithm with running time $O^*(2^{n/2}) = O(1.4143^n)$ [KMRR05].

Theorem 16. *The described algorithm solves $\#$ DOMINATING SET in $3^{n/5+o(n)}$ time and polynomial space on subcubic graphs.*

A part of this improvement is due our new 3-way branching. It is inspired by the inclusion/exclusion branching of [vRNvD09], but to the best of our knowledge, it has not been used in this form before. The algorithm of [KMRR05] used a 4-way branching, giving a running time of $O^*(4^{n/4}) = O^*(2^{n/2})$. If we use their analysis, or the analysis of [SS07], with our 3-way branching, one obtains a running time bound of $O^*(3^{n/4}) = O(1.3161^n)$. Adding our separator-based choice of the pivot vertices improves the running time further to $3^{n/5+o(n)} = O(1.2458^n)$.

5.2. General graphs. As another application of our Separate, Measure and Conquer technique, we show that by changing the preference for the pivot vertex in the subcubic subproblem, taking advantage of a separator, the running time of van Rooij's algorithm [vR10] improves from $O(1.5673^n)$ to $O(1.5183^n)$.

Let us recall van Rooij's algorithm #SC-vR [vR10] (see Algorithm 1). It first reduces the #DOMINATING SET problem to the #SET COVER problem, where each vertex corresponds to an element of the universe \mathcal{U} and the closed neighborhood of each vertex corresponds to a set $X \in \mathcal{S}$. Then, there is a cardinality-preserving bijection between dominating sets of the graph and set covers of the instance $(\mathcal{U}, \mathcal{S})$. The *incidence graph* of $(\mathcal{U}, \mathcal{S})$ is the bipartite graph $(\mathcal{S} \cup \mathcal{U}, E)$ which has an edge $(X, e) \in E$ between a set $X \in \mathcal{S}$ and an element $e \in \mathcal{U}$ if and only if $e \in X$. In the #SET COVER problem, the input is an incidence graph of an instance $(\mathcal{U}, \mathcal{S})$, and the task is to compute for each size κ , $0 \leq \kappa \leq |\mathcal{S}|$, the number of set covers of size κ . As an additional input, the algorithm has a set A of annotated vertices that is initially empty. When the algorithm finds a vertex of degree at most 1 or a vertex of degree 2 that has the same neighborhood as some other vertex, it annotates this vertex, which means the vertex is effectively removed from the instance, and is left to a polynomial-time dynamic programming algorithm, #SC-DP, that handles instances of maximum degree at most 2 plus annotated vertices.

Van Rooij showed that this algorithm has running time $O(1.5673^n)$. We will now modify it to obtain Algorithm #SC (see Algorithm 2), which takes advantage of small separators in subcubic graphs. Then, we will show that this modification improves the running time to $O(1.5183^n)$. Algorithmically, the change is simple. The parts that remain essentially unchanged (except that the separation is passed along in the recursive calls) are written in gray in Algorithm 2. When the algorithm reaches an instance with maximum degree at most 3, it computes a separator of the incidence graph and prefers to branch on vertices in the separator. To do this, we add to the input a separation (L, S, R) of $I - A$. Initially, this separation is $(\emptyset, \emptyset, V(I) - A)$. We note that a separation (L, S, R) for $I - A$ can easily be transformed into a separation (L', S', R') for I . For this, we assign each vertex from A to one of its neighbors in $V(I) \setminus A$ when it is annotated (or to an arbitrary vertex if it has degree 0). A vertex from A will then be in L' , S' , or R' if it is assigned to a vertex in L , S , or R , respectively. The vertices in A will not affect the measure, defined momentarily. Our algorithm also explicitly handles connected components when the graph is not connected, and it calls a subroutine, #3SC (Algorithm 3), when the maximum degree is 3.

We are now ready to upper bound the running time of Algorithm #SC (Algorithm 2). For instances with maximum degree at least 4, we use the same measure as van Rooij,

$$\mu_4 = \sum_{e \in \mathcal{U} \setminus A} w_{\text{elt}}(d_{I-A}(e)) + \sum_{X \in \mathcal{S} \setminus A} w_{\text{set}}(d_{I-A}(X)) .$$

Here, $w_{\text{elt}}(\cdot)$ and $w_{\text{set}}(\cdot)$ are non-negative real functions of the vertex degrees, which will be determined later. For instances with maximum degree 3, we use the following measure,

$$\mu_3 = \mu_s(S) + \mu_r(R) + \max \left(0, B - \frac{\mu_r(R) - \mu_r(L)}{2} \right) + (1 + B) \cdot \log_{1+\epsilon}(\mu_r(R) + \mu_s(S)),$$

Algorithm 1: #SC-vR(I,A) – van Rooij’s algorithm for #SET COVER.

Input: The incidence graph $I = (\mathcal{S} \cup \mathcal{U}, E)$ of \mathcal{S} and a set of annotated vertices A
Output: A list containing the number of set covers of \mathcal{S} of each cardinality κ

```

1 if there exists a vertex  $v \in \mathcal{S} \cup \mathcal{U}$  of degree at most one in  $I - A$  then
2   return #SC-vR( $I, A \cup \{v\}$ )
3 else if there exist two vertices  $v_1, v_2 \in \mathcal{S} \cup \mathcal{U}$  both of degree two in  $I - A$  that have the same
   two neighbors then
4   return #SC-vR( $I, A \cup \{v_1\}$ )
5 else
6   Let  $X \in \mathcal{S}$  and  $e \in \mathcal{U}$  be a set vertex and an element vertex, each of maximum degree in
    $I - A$ 
7   if  $d_{I-A}(X) \leq 2$  and  $d_{I-A}(e) \leq 2$  then
8     return #SC-DP( $I$ )
9   else if  $d_{I-A}(X) > d_{I-A}(e)$  then
10    Let  $L_{\text{take}} = \text{\#SC-vR}(I - N_I[s], A \setminus N_I(s))$  and increase all cardinalities by one
11    Let  $L_{\text{discard}} = \text{\#SC-vR}(I - s, A)$ 
12    return  $L_{\text{take}} + L_{\text{discard}}$ 
13  else
14    Let  $L_{\text{optional}} = \text{\#SC-vR}(I - e, A)$ 
15    Let  $L_{\text{forbidden}} = \text{\#SC-vR}(I - N_I[e], A \setminus N_I(e))$ 
16    return  $L_{\text{optional}} - L_{\text{forbidden}}$ 

```

where

$$\begin{aligned} \mu_s(S) &= \sum_{s \in S} w_{\text{sep}}(d_{I-A}(s)), \\ \mu_r(R) &= \sum_{r \in R} w_{\text{right}}(d_{I-A}(r)), \text{ and} \\ B &= 6 \cdot w_{\text{right}}(3). \end{aligned}$$

Again, $w_{\text{sep}}(\cdot)$ and $w_{\text{right}}(\cdot)$ are non-negative real functions of the vertex degrees, which will be determined later. We commonly refer to $w_{\text{elt}}(i)$, $w_{\text{set}}(i)$, $w_{\text{sep}}(i)$, $w_{\text{right}}(i)$ as the weight of an element vertex, a set vertex, a separator vertex and a right vertex, respectively, of degree i .

To combine the analysis for subcubic instances and instances with maximum degree at least 4 using Lemma 15, we impose the following constraints, guaranteeing that $\mu_4 \geq \mu_3$ when $\mu_4 = \omega(1)$. (When $\mu_4 = O(1)$, the algorithm will take constant time.)

$$w_{\text{elt}}(3) > w_{\text{right}}(3) \qquad w_{\text{set}}(3) > w_{\text{right}}(3) \tag{31}$$

$$w_{\text{elt}}(i) \geq w_{\text{right}}(i), \text{ and } w_{\text{set}}(i) \geq w_{\text{right}}(i), \qquad i \in \{0, 1, 2\}. \tag{32}$$

For instances with degree at least 4, we obtain exactly the same constraints on the measure as in [vR10]. Denoting $\Delta w_{\text{elt}}(i) = w_{\text{elt}}(i) - w_{\text{elt}}(i-1)$ and $\Delta w_{\text{set}}(i) = w_{\text{set}}(i) - w_{\text{set}}(i-1)$, these

Algorithm 2: #SC(I,A,S) – separator-based #SET COVER algorithm. (Lines shown in grey are essentially unchanged from Algorithm 1.)

Input: The incidence graph $I = (\mathcal{S} \cup \mathcal{U}, E)$ of \mathcal{S} , a set of annotated vertices A , and a separation (L, S, R) of $I - A$

Output: A list containing the number of set covers of \mathcal{S} of each cardinality κ

```

1 if  $I$  has a connected component  $X \subsetneq V(I)$  then
2   Let  $L_X = \text{\#SC}(I[X], A \cap X, (\emptyset, \emptyset, R))$ 
3   Let  $L_{\overline{X}} = \text{\#SC}(I - X, A \setminus X, (\emptyset, \emptyset, V(I) \setminus X))$ 
4   return  $L$  where  $L[i] = \sum_{j=0}^i L_X[j] \cdot L_{\overline{X}}[i-j]$ 
5 else if there exists a vertex  $v \in V(I)$  of degree at most one in  $I - A$  then
6   return  $\text{\#SC}(I, A \cup \{v\}, (L \setminus \{v\}, S \setminus \{v\}, R \setminus \{v\}))$ 
7 else if there exist two vertices  $v_1, v_2 \in V(I) \setminus A$  both of degree two in  $I - A$  that have the
   same two neighbors then
8   return  $\text{\#SC}(I, A \cup \{v_1\}, (L \setminus \{v_1\}, S \setminus \{v_1\}, R \setminus \{v_1\}))$ 
9 else
10   Let  $X \in \mathcal{S}$  and  $e \in \mathcal{U}$  be a set vertex and an element vertex, each of maximum degree in
       $I - A$ 
11   if  $d_{I-A}(X) \leq 2$  and  $d_{I-A}(e) \leq 2$  then
12     return  $\text{\#SC-DP}(I)$ 
13   else if  $d_{I-A}(X) \leq 3$  and  $d_{I-A}(e) \leq 3$  then
14     return  $\text{\#3SC}(I, A, S)$ 
15   else if  $d_{I-A}(X) > d_{I-A}(e)$  then
16     Let  $L_{\text{take}} = \text{\#SC}(I - N_I[s], A \setminus N_I(s), (\emptyset, \emptyset, V(I) \setminus N_I[s]))$  and increase all cardinalities
        by one
17     Let  $L_{\text{discard}} = \text{\#SC}(I - s, A, (\emptyset, \emptyset, V(I) \setminus \{s\}))$ 
18     return  $L_{\text{take}} + L_{\text{discard}}$ 
19   else
20     Let  $L_{\text{optional}} = \text{\#SC}(I - e, A, (\emptyset, \emptyset, V(I) \setminus \{e\}))$ 
21     Let  $L_{\text{forbidden}} = \text{\#SC}(I - N_I[e], A \setminus N_I(e), (\emptyset, \emptyset, V(I) \setminus N_I[e]))$ 
22     return  $L_{\text{optional}} - L_{\text{forbidden}}$ 

```

constraints are

$$w_{\text{elt}}(0) = w_{\text{elt}}(1) = 0, \quad w_{\text{set}}(0) = w_{\text{set}}(1) = 0, \quad (33)$$

$$\Delta w_{\text{elt}}(i) \geq 0, \quad \Delta w_{\text{set}}(i) \geq 0 \quad \text{for all } i \geq 2, \quad (34)$$

$$\Delta w_{\text{elt}}(i) \geq \Delta w_{\text{elt}}(i+1), \quad \Delta w_{\text{set}}(i) \geq \Delta w_{\text{set}}(i+1) \quad \text{for all } i \geq 2, \quad (35)$$

$$2 \cdot \Delta w_{\text{elt}}(3) \leq w_{\text{elt}}(2), \text{ and } 2 \cdot \Delta w_{\text{set}}(4) \leq w_{\text{set}}(2). \quad (36)$$

For branching on a vertex with degree $d \geq 4$ with r_i neighbors of degree i in $I - A$, where $\sum_{i=2}^d r_i = d$, we have the following constraints,

$$2^{-w_{\text{set}}(d) - \sum_{i=2}^{\infty} r_i \cdot w_{\text{elt}}(i) - \Delta w_{\text{set}}(d) \cdot \sum_{i=2}^{\infty} r_i \cdot (i-1)} + 2^{-w_{\text{set}}(d) - \sum_{i=2}^{\infty} r_i \cdot \Delta w_{\text{elt}}(i)} \leq 1 \quad (37)$$

Algorithm 3: #3SC(I,A,S) – separator-based #SET COVER algorithm for subcubic instances.

Input: The incidence graph $I = (\mathcal{S} \cup \mathcal{U}, E)$ of \mathcal{S} , a set of annotated vertices A , and a separation (L, S, R) of $I - A$

Output: A list containing the number of set covers of \mathcal{S} of each cardinality κ

```

1 if  $S = \emptyset$  then
2    $\lfloor$  Compute a balanced separation  $(L, S, R)$  with respect to the measure  $\mu_r$  using Lemma 14
3 if  $\mu_r(L) > \mu_r(R)$  then
4    $\lfloor$  Swap  $L$  and  $R$ 
5 if there exists a vertex  $s \in S$  with no neighbor in  $L$  then
6    $\lfloor$  return #SC( $I, A, (L, S \setminus \{s\}, R \cup \{s\})$ )
7 else if there exists a vertex  $s \in S$  with no neighbor in  $R$  then
8    $\lfloor$  return #SC( $I, A, (L \cup \{s\}, S \setminus \{s\}, R)$ )
9 else if there exists a vertex  $s \in S \setminus A$  with  $d_{I-A}(s) = 2$  then
10   $\lfloor$  if  $(L, S, R)$  is balanced then
11     $\lfloor$  Let  $l$  be the first degree-3 vertex or vertex from  $S$  encountered when moving from  $s$  to
12     $\lfloor$  the left along a path  $P$  of degree-2 vertices in  $I - A$ 
13     $\lfloor$  return #SC( $I, A, (L \setminus (P \cup \{l\}), (S \setminus \{s\}) \cup \{l\}, R \cup P \cup \{s\})$ )
14   $\lfloor$  else
15     $\lfloor$  Let  $r$  be the first degree-3 vertex or vertex from  $S$  encountered when moving from  $s$  to
16     $\lfloor$  the right along a path  $P$  of degree-2 vertices in  $I - A$ 
17     $\lfloor$  return #SC( $I, A, (L \cup P \cup \{s\}, (S \setminus \{s\}) \cup \{l\}, R \setminus (P \cup \{l\}))$ )
18 else if  $\mu_r(R) - \mu_r(L) > B$  and there exists a vertex  $s \in S$  with two neighbors in  $L$  and one
19 neighbor  $r$  in  $R$  that has degree 3 in  $I - A$  then
20    $\lfloor$  return #SC( $I, A, (L \cup \{s\}, (S \cup \{r\}) \setminus \{s\}, R \setminus \{r\})$ )
21 else if  $\mu_r(R) - \mu_r(L) > B$  and there exists a vertex  $s \in S$  with two neighbors in  $L$  and one
22 neighbor  $r$  in  $R$  with  $N_{I-A}(r) = \{s, s'\}$  for some  $s' \in S$  then
23    $\lfloor$  return #SC( $I, A, (L \cup \{s, r\}, S \setminus \{s\}, R \setminus \{r\})$ )
24 else if there exists an element  $s \in \mathcal{U} \cap S$  then
25    $\lfloor$  Let  $L_{\text{optional}} = \text{\#SC}(I - s, A, (L, S \setminus \{s\}, R))$ 
26    $\lfloor$  Let  $L_{\text{forbidden}} = \text{\#SC}(I - N_I[s], A \setminus N_I(s), (L \setminus N(s), S \setminus N[s], R \setminus N[s]))$ 
27    $\lfloor$  return  $L_{\text{optional}} - L_{\text{forbidden}}$ 
28 else
29    $\lfloor$  Let  $s \in \mathcal{S} \cap S$  be a set in  $S$ 
30    $\lfloor$  Let  $L_{\text{discard}} = \text{\#SC}(I - s, A, (L, S \setminus \{s\}, R))$ 
31    $\lfloor$  Let  $L_{\text{take}} = \text{\#SC}(I - N_I[s], A \setminus N_I(s), (L \setminus N(s), S \setminus N[s], R \setminus N[s]))$  and increase all
32    $\lfloor$  cardinalities by one
33    $\lfloor$  return  $L_{\text{take}} + L_{\text{discard}}$ 

```

if the vertex is a set vertex, and

$$2^{-w_{\text{elt}}(d) - \sum_{i=2}^{\infty} r_i \cdot w_{\text{set}}(i) - \Delta w_{\text{elt}}(d) \cdot \sum_{i=2}^{\infty} r_i \cdot (i-1)} + 2^{-w_{\text{elt}}(d) - \sum_{i=2}^{\infty} r_i \cdot \Delta w_{\text{set}}(i)} \leq 1 \quad (38)$$

if it is an element vertex. We note that the constraints (37) have $r_d = 0$ since the algorithm prefers to branch on elements when there is both an element and a set of maximum degree at least 4. Constraints (33)–(38) are directly taken from [vR10].

For instances where $I - A$ has maximum degree 3, let us first define the minimum decrease in measure due to a decrease in the degree in $I - A$ of a vertex of degree 2 or 3 in a balanced or imbalanced instance:

$$\delta_{\text{deg-dec}} = \min_{i \in \{2,3\}} \left\{ w_{\text{sep}}(i) - w_{\text{sep}}(i-1), \frac{w_{\text{right}}(i) - w_{\text{right}}(i-1)}{2} \right\}. \quad (39)$$

To make sure that annotating vertices does not increase the measure, we require that the decrease of the degree of a vertex (half-edge deletion) does not increase the measure:

$$\delta_{\text{deg-dec}} \geq 0. \quad (40)$$

For computing a new separator in line 2, constraint (27) becomes

$$\begin{aligned} w_{\text{sep}}(3)/6 + 5/12 \cdot w_{\text{right}}(3) &< w_{\text{right}}(3), \text{ or} \\ w_{\text{sep}}(3) &< 7/2 \cdot w_{\text{right}}(3). \end{aligned} \quad (41)$$

For dragging a separator vertex into R if it has no neighbor in L (line 6), imbalanced instances are most constraining:

$$-w_{\text{sep}}(d) + w_{\text{right}}(d) \leq 0, \quad 2 \leq d \leq 3. \quad (42)$$

For dragging a separator vertex into L if it has no neighbor in R (line 8), balanced instances are most constraining, imposing the constraints

$$-w_{\text{sep}}(d) + 1/2 \cdot w_{\text{right}}(d) \leq 0, \quad 2 \leq d \leq 3,$$

which are no more constraining than (42).

If there is a vertex $s \in S$ with $d_{I-A}(s) = 2$ (line 9), then s is dragged – along with any other degree-2 vertices that are dragged into S – to the right if the instance is balanced and to the left if the instance is imbalanced. In both cases we obtain the constraints

$$-w_{\text{sep}}(2) + w_{\text{sep}}(3) + 1/2 \cdot (w_{\text{right}}(2) - w_{\text{right}}(3)) \leq 0. \quad (43)$$

For the operation in line 17, where $\mu_r(R) - \mu_r(L) > B$ and some vertex $s \in S$ has two neighbors in L and one degree-3 neighbor in R , the vertex s is dragged to the left. In the worst case, the resulting instance is balanced, imposing the following constraints on the measure,

$$-w_{\text{sep}}(3) + w_{\text{sep}}(3) + 1/2 \cdot (w_{\text{right}}(3) - w_{\text{right}}(3)) \leq 0,$$

which always holds.

For the operation in line 19, where $\mu_r(R) - \mu_r(L) > B$ and some vertex $s \in S$ has two neighbors in L and one degree-2 neighbor r in R that has another neighbor s' in S , the vertices s and r are dragged to the left. In the worst case, the resulting instance is balanced, imposing the following constraints on the measure,

$$-w_{\text{sep}}(3) + 1/2 \cdot w_{\text{right}}(3) \leq 0,$$

which holds due to (42).

For the constraints of the branching steps, we will take into account the decrease of vertex degrees in the second neighborhood of a set that we add to the set cover and of an element that we forbid to cover. The analysis will be simplified by imposing the following constraints,

$$2 \cdot \Delta w_{\text{sep}}(3) \leq w_{\text{sep}}(2) \quad 2 \cdot \Delta w_{\text{right}}(3) \leq w_{\text{right}}(2). \quad (44)$$

They will enable us to account for a decrease in measure of $\delta_{\text{deg-dec}}$ for each edge that leaves the second neighborhood, even when two of these edges are incident to one vertex of degree 2. We

cannot relax these equations for sets as in (36): When $I - A$ has maximum degree 3 and it contains an element of degree 3, then van Rooij's algorithm prefers to branch on a degree-3 element, whereas the new algorithm cannot guarantee to find a degree-3 element in S .

For the two branching rules (lines 20–23 and 25–28), the algorithm selects a separator vertex $s \in S$, which has degree 3 in $I - A$ due to line 9. In the first branch, s is deleted from the graph, and in the second branch, $N_I[s]$ is deleted.

First, consider the case where s has a neighbor s' in S . By lines 5 and 7, $N_{I-A}(s) = \{l, s', r\}$ and $N_{I-A}(s') = \{l', s, r'\}$ with $\{l, l'\} \subseteq L$ and $\{r, r'\} \subseteq R$. Since I is bipartite, we have that $l \neq l'$ and $r \neq r'$. In the balanced case, the branch where only s is deleted has a decrease in measure of $w_{\text{sep}}(3)$ for s , $\Delta w_{\text{sep}}(3)$ for s' , $1/2 \cdot \Delta w_{\text{right}}(d_{I-A}(r))$ for r , and $1/2 \cdot \Delta w_{\text{right}}(d_{I-A}(l))$ for l , and the branch where $N_{I-A}[s]$ is deleted has a decrease in measure of $2w_{\text{sep}}(3)$ for s and s' , $1/2 \cdot w_{\text{right}}(d_{I-A}(r))$ for r , $1/2 \cdot w_{\text{right}}(d_{I-A}(l))$ for l , and $(d_{I-A}(r) + d_{I-A}(l)) \cdot \delta_{\text{deg-dec}}$ for the vertices in $N_{I-A}(N_{I-A}[s])$. Thus, we get the following constraints for the balanced case,

$$\begin{aligned} & 2^{-w_{\text{sep}}(3) - \Delta w_{\text{sep}}(3) - 1/2 \cdot (\Delta w_{\text{right}}(d_r) + \Delta w_{\text{right}}(d_l))} \\ & + 2^{-2w_{\text{sep}}(3) - 1/2 \cdot (w_{\text{right}}(d_r) + w_{\text{right}}(d_l)) - (d_r + d_l) \cdot \delta_{\text{deg-dec}}} \\ & \leq 1, \end{aligned} \quad 2 \leq d_l, d_r \leq 3. \quad (45)$$

In the imbalanced case, we obtain the following set of constraints, which are no more constraining than the previous set,

$$\begin{aligned} & 2^{-w_{\text{sep}}(3) - \Delta w_{\text{sep}}(3) - \Delta w_{\text{right}}(d_r)} \\ & + 2^{-2w_{\text{sep}}(3) - w_{\text{right}}(d_r) - d_r \cdot \delta_{\text{deg-dec}}} \\ & \leq 1, \end{aligned} \quad 2 \leq d_r \leq 3.$$

Here, the number of vertices in $N_{I-A}(N_{I-A}[s]) \cap (S \cup R)$ is d_r , accounting for the vertex r' and the neighbors of r besides s .

Now, consider the case where s has two neighbors in L or in R . Since the graph is bipartite, these two neighbors are not adjacent. If $\mu_r(R) - \mu_r(L) \leq B$, then the two created subinstances are balanced, which gives us the following set of constraints:

$$\begin{aligned} & 2^{-w_{\text{sep}}(3) - 1/2 \cdot (\Delta w_{\text{right}}(d_1) + \Delta w_{\text{right}}(d_2) + \Delta w_{\text{right}}(d_3))} \\ & + 2^{-w_{\text{sep}}(3) - 1/2 \cdot (w_{\text{right}}(d_1) + w_{\text{right}}(d_2) + w_{\text{right}}(d_3)) - (d_1 + d_2 + d_3 - 3) \cdot \delta_{\text{deg-dec}}} \\ & \leq 1, \end{aligned} \quad 2 \leq d_1, d_2, d_3 \leq 3. \quad (46)$$

Otherwise, $\mu_r(R) - \mu_r(L) > B$. If s has two neighbors in R , then the worst case is the balanced one, which is covered by (46). Finally, if s has two neighbors in L , then its neighbor $r \in R$ has $d_{I-A}(r) = 2$ due to line 16. The imbalanced case is not covered by (46) and incurs the following constraint on the measure.

$$\begin{aligned} & 2 \cdot 2^{-w_{\text{sep}}(3) - w_{\text{right}}(2) - \Delta w_{\text{right}}(d)} \\ & \leq 1, \end{aligned} \quad 2 \leq d \leq 3. \quad (47)$$

Note that the deletion of s triggers that r is removed by line 6 in Algorithm 2, which decreases the degree of its other neighbor, which is in R due to line 18.

This gives us all the constraints on μ_3 for the analysis. To obtain values for the various weights, we set

$$w_{\text{elt}}(i) = w_{\text{elt}}(i + 1), \quad w_{\text{set}}(i) = w_{\text{set}}(i + 1), \quad i \geq 6, \quad (48)$$

and we minimize $w_{\text{set}}(6) + w_{\text{elt}}(6)$. Solving this convex program gives the following values for the weights:

$w_{\text{right}}(2) = 0.15282$	$w_{\text{elt}}(2) = 0.15384$	$w_{\text{set}}(2) = 0.16408$
$w_{\text{right}}(3) = 0.22669$	$w_{\text{elt}}(3) = 0.22732$	$w_{\text{set}}(3) = 0.24592$
$w_{\text{sep}}(2) = 0.75630$	$w_{\text{elt}}(4) = 0.26684$	$w_{\text{set}}(4) = 0.29320$
$w_{\text{sep}}(3) = 0.78943$	$w_{\text{elt}}(5) = 0.29023$	$w_{\text{set}}(5) = 0.30224$
	$w_{\text{elt}}(6) = 0.30019$	$w_{\text{set}}(6) = 0.30224$

For subcubic instances, the depth of the search tree is bounded by a polynomial since each recursive call decreases the measure $(|S_2| + |S|) \cdot \frac{\mu_r(V(I))}{w_{\text{right}}(2)} + \left| \frac{\mu_r(R) - \mu_r(L)}{w_{\text{right}}(2)} \right|$ by at least 1, where S_2 is the set of vertices in S that have degree 2 in $I - A$. By Lemma 5, we conclude that subcubic instances are solved in time $O^*(2^{\mu_3})$. Now, using Lemma 15 with $\mu = \mu_4$, $\mu' = \mu_3$ and $\eta = |V(I) \setminus A|$, we conclude that the running time of the algorithm is upper bounded by $O^*(2^{\mu_4})$. Thus, the algorithm solves #DOMINATING SET in time $O^*(2^{(w_{\text{elt}}(6) + w_{\text{set}}(6)) \cdot n}) = O^*(1.5183^n)$, where n is the number of vertices.

Theorem 17. *Algorithm #SC solves #SET COVER in time $O^*(2^{0.30019 \cdot |U| + 0.30224 \cdot |S|})$ and #DOMINATING SET in time $O(1.5183^n)$ and polynomial space.*

5.3. Comments on the #SC algorithm and its analysis. Let us make a few final comments on the algorithm and its analysis. First, while the algorithm of van Rooij always prefers to branch on an element when the graph contains both an element and a set of maximum degree, this is not always possible when the algorithm needs to branch on vertices in the separator. However, this disadvantage is overwhelmed by the gain due to the separator branching.

Second, a certain amount of care needs to be taken to make sure that the algorithm terminates. For example, if we had omitted “that has degree 3 in $I - A$ ” in line 16 of Algorithm 3, then lines 12 and 17 could have eternally dragged vertices alternately to the left and to the right.

Third, the initial SET COVER transformation is not absolutely necessary, since we may equivalently label the vertices as in Subsection 5.1. We can then assign a weight to each vertex which depends on its label and the number of neighbors with each label. However, our attempt resulted in an unmanageable number of constraints.⁴ We tried several compromises to simplify the analysis (merge labels, simplify the degree-spectrum), and we found that, of these, the simplification corresponding exactly to the SET COVER translation performed best. Here, a graph vertex u corresponds to an element e , which encodes that the vertex needs to be dominated, and to a set X , which encodes that it can be in the dominating set. Thus, in the measure, the weight of u is the weight of e plus the weight of X . Therefore, the weight of u depends on the degree of e and the degree of X , but not on the label of u and the degree of u . It would be interesting to know whether this choice of weights compromises the optimality of the analysis.

Finally, we remark that it did not help the analysis to allow different weights for sets and elements in the subcubic analysis. To further improve the running time, it will not be sufficient to improve the subcubic case, since none of the degree-3 constraints turn out tight.

⁴With 8Gb of memory, we were only able to handle around 1.25 million constraints using a 64-bit version of AMPL and the solver NPSOL, and such instances were solved in around 10 minutes. This was not even sufficient to analyze running times for degree-4 instances.

6. LIMITATIONS OF THE METHOD

In this section, we discuss when we can and cannot expect to achieve improved running times using the Separate, Measure and Conquer method. The most important feature an algorithm needs to have is that it eventually encounters instances where small balanced separators can be computed efficiently. This is the case for algorithms that branch on graphs of bounded degree in their final stages, but other means of obtaining graphs with relatively small separators are conceivable. This might include cases where the treewidth of the graph is bounded by a fraction of the number of vertices, where the instance has a small backdoor set to bounded treewidth instances [GS13], and graphs with small treewidth modulators [KLP⁺13, FLMS12].

The first limitation of our method occurs when a non-separator based algorithm is already so fast that branching on separators does not add an advantage. For example, the current fastest algorithm for MAXIMUM INDEPENDENT SET on subcubic graphs has worst-case running time $O(1.0836^n)$, where n is the number of vertices of the input graph [XN13]. Merely branching on all the vertices of the separator would take time $2^{n/6+o(n)}$ and $2^{1/6} \approx 1.1225 > 1.0836$. This full time will be required, e.g., if the computed separator happens to be an independent set, so that a decision for one vertex of the separator (put it in the independent set or not) does not have any direct implications for the other vertices in the separator. While this is a limitation of our method, it also motivates the study of balanced separators with additional properties. A small but relatively dense separator could be useful for MAXIMUM INDEPENDENT SET branching algorithms.

The second limitation is that our Separate, Measure and Conquer subroutines can often be replaced by treewidth-based subroutines [FGSS09], leading to smaller worst-case running times but exponential space usage. In fact, replacing the branching on graphs with small maximum degree by a treewidth-based dynamic programming algorithm, such as [vRBR09], is a well-known method [FGSS09] to improve the worst-case running time of an algorithm at the expense of exponential-space usage. For instance, the currently fastest exponential-space algorithm for #DOMINATING SET [NvRvDXX] uses a treewidth-based subroutine on bounded degree instances to improve the running time. It solves #DOMINATING SET in $O(1.5002^n)$ time and exponential space. Using only polynomial space is however recognized as a significant advantage, especially for algorithms that may run much faster on real-world instances than their worst-case running time bounds, and the field devotes attention to both categories. An exponential space usage would quickly become a bottleneck in the execution of the algorithm.

7. CONCLUSION

We have presented a new method to analyze separator based branching algorithms within the Measure and Conquer framework. It uses a novel kind of measure that is able to take advantage of a global structure in the instance and amortize a sudden large gain, due to the instance decomposing into several independent subinstances, over a linear number of previous branchings.

This new method will provide both opportunities and challenges in the design and analysis of other exponential-time branching algorithms – opportunities because we believe it is widely applicable, and challenges because it complicates the analysis and leads to more choices in the design of algorithms. As usual, finding a set of reductions leading to a small running time upper bound in a Measure and Conquer analysis remains an art.

Finally, our analysis might open the way for other measures relying on global structures of instances. One intriguing question in this respect is how to take advantage of the knowledge that certain branching rules can only be applied a small fraction of times on the path from the root to a leaf of a search tree.

ACKNOWLEDGMENTS

The research was supported in part by the DIMACS 2006–2010 Special Focus on Discrete Random Systems, NSF grant DMS-0602942. Serge Gaspers is the recipient of an Australian Research Council Discovery Early Career Researcher Award (project number DE120101761). NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

REFERENCES

- [AHI05] Michael Alekhnovich, Edward A. Hirsch, and Dmitry Itsykson, *Exponential lower bounds for the running time of dpll algorithms on satisfiable formulas*, J. Autom. Reasoning **35** (2005), no. 1-3, 51–72.
- [AS00] Dimitris Achlioptas and Gregory B. Sorkin, *Optimal myopic algorithms for random 3-SAT*, 41st Annual Symposium on Foundations of Computer Science, IEEE Comput. Soc. Press, Los Alamitos, CA, 2000, pp. 590–600.
- [BEM⁺04] Sergei L. Bezrukov, Robert Elsässer, Burkhard Monien, Robert Preis, and Jean-Pierre Tillich, *New spectral lower bounds on the bisection width of graphs*, Theoretical Computer Science **320** (2004), no. 2-3, 155–174.
- [BK96] Hans L. Bodlaender and Ton Kloks, *Efficient and constructive algorithms for the pathwidth and treewidth of graphs*, Journal of Algorithms **21** (1996), no. 2, 358–402.
- [Bod98] Hans L. Bodlaender, *A partial k -arboretum of graphs with bounded treewidth*, Theoretical Computer Science **209** (1998), no. 1-2, 1–45.
- [BS06] Armin Biere and Carsten Sinz, *Decomposing SAT problems into connected components*, JSAT, Journal on Satisfiability, Boolean Modeling and Computation **2** (2006), no. 1-4, 201–208.
- [DM07] Rina Dechter and Robert Mateescu, *And/or search spaces for graphical models*, Artificial Intelligence **171** (2007), no. 2-3, 73–106.
- [FGK09] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch, *A measure & conquer approach for the analysis of exact algorithms*, Journal of the ACM **56** (2009), no. 5.
- [FGSS09] Fedor V. Fomin, Serge Gaspers, Saket Saurabh, and Alexey A. Stepanov, *On two techniques of combining branching and treewidth*, Algorithmica **54** (2009), no. 2, 181–207.
- [FH06] Fedor V. Fomin and Kjartan Høie, *Pathwidth of cubic graphs and exact algorithms*, Information Processing Letters **97** (2006), no. 5, 191–196.
- [FLMS12] Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, and Saket Saurabh, *Planar F -deletion: Approximation, kernelization and optimal fpt algorithms*, Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012), IEEE Computer Society, 2012, pp. 470–479.
- [FQ85] Eugene C. Freuder and Michael J. Quinn, *Taking advantage of stable sets of variables in constraint satisfaction problems*, Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI 1985), Morgan Kaufmann, 1985, pp. 1076–1078.
- [Gas10] Serge Gaspers, *Exponential time algorithms - structures, measures, and bounds*, VDM, 2010.
- [GK14] Alexander Golovnev and Konstantin Kutkov, *New exact algorithms for the 2-constraint satisfaction problem*, Theoretical Computer Science **526** (2014), 18–27.
- [GLS00] Georg Gottlob, Nicola Leone, and Francesco Scarcello, *A comparison of structural CSP decomposition methods*, Artificial Intelligence **124** (2000), no. 2, 243–282.
- [GS12] Serge Gaspers and Gregory B. Sorkin, *A universally fastest algorithm for Max 2-Sat, Max 2-CSP, and everything in between*, J. Comput. System Sci. **78** (2012), no. 1, 305–335. MR 2896365
- [GS13] Serge Gaspers and Stefan Szeider, *Strong backdoors to bounded treewidth SAT*, Proceedings of the 54th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2013), IEEE Computer Society, 2013, pp. 489–498.
- [Iwa12] Yoichi Iwata, *A faster algorithm for dominating set analyzed by the potential method*, Proceedings of the 6th International Symposium on Parameterized and Exact Computation (IPEC 2011), Lecture Notes in Computer Science, vol. 7112, Springer, 2012, pp. 41–54.
- [KLP⁺13] Eun Jung Kim, Alexander Langer, Christophe Paul, Felix Reidl, Peter Rossmanith, Ignasi Sau, and Somnath Sikdar, *Linear kernels and single-exponential algorithms via protrusion decompositions*, Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP 2013), Lecture Notes in Computer Science, vol. 7965, Springer, 2013, pp. 613–624.

- [KMRR05] Joachim Kneis, Daniel Mölle, Stefan Richter, and Peter Rossmanith, *Algorithms based on the treewidth of sparse graphs*, Proceedings of the 31st International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2005), Lecture Notes in Computer Science, vol. 3787, Springer, Berlin, 2005, pp. 385–396.
- [LT77] Richard J. Lipton and Robert Endre Tarjan, *Application of a planar separator theorem*, Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS 1977), IEEE Computer Society, 1977, pp. 162–170.
- [LvB04] Wei Li and Peter van Beek, *Guiding real-world SAT solving with dynamic hypergraph separator decomposition*, Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004), IEEE Computer Society, 2004, pp. 542–548.
- [MP01] Burkhard Monien and Robert Preis, *Upper bounds on the bisection width of 3- and 4-regular graphs*, Proceedings of the 26th International Symposium on Mathematical Foundations of Computer Science (MFCS 2001), Lecture Notes in Computer Science, vol. 2136, Springer, 2001, pp. 524–536.
- [MP06] Burkhard Monien and Robert Preis, *Upper bounds on the bisection width of 3- and 4-regular graphs*, J. Discrete Algorithms **4** (2006), no. 3, 475–498. MR 2258338 (2008f:05159)
- [NvRvDXX] Jesper Nederlof, Johan M. M. van Rooij, and Thomas C. van Dijk, *Inclusion/exclusion meets measure and conquer*, Algorithmica (XX), to appear.
- [RS86] Neil Robertson and Paul D. Seymour, *Graph minors. II. Algorithmic aspects of tree-width*, Journal of Algorithms **7** (1986), no. 3, 309–322.
- [SS06] Alexander D. Scott and Gregory B. Sorkin, *Solving sparse random instances of Max Cut and Max 2-CSP in linear expected time*, Comb. Probab. Comput. **15** (2006), no. 1-2, 281–315.
- [SS07] ———, *Linear-programming design and analysis of fast algorithms for Max 2-CSP*, Discrete Optimization **4** (2007), no. 3-4, 260–287.
- [SS09] Alexander D. Scott and Gregory B. Sorkin, *Polynomial constraint satisfaction problems, graph bisection, and the Ising partition function*, ACM Trans. Algorithms **5** (2009), no. 4, Art. 45, 27.
- [vR10] Johan M. M. van Rooij, *Polynomial space algorithms for counting dominating sets and the domatic number*, Proceedings of the 7th International Conference on Algorithms and Complexity (CIAC 2010), Lecture Notes in Computer Science, vol. 6078, Springer, 2010, pp. 73–84.
- [vRB11] Johan M. M. van Rooij and Hans L. Bodlaender, *Exact algorithms for dominating set*, Discrete Applied Mathematics **159** (2011), no. 17, 2147–2164.
- [vRBR09] Johan M. M. van Rooij, Hans L. Bodlaender, and Peter Rossmanith, *Dynamic programming on tree decompositions using generalised fast subset convolution*, Proceedings of the 17th Annual European Symposium on Algorithms (ESA 2009), Lecture Notes in Computer Science, vol. 5757, Springer, 2009, pp. 566–577.
- [vRNvD09] Johan M. M. van Rooij, Jesper Nederlof, and Thomas C. van Dijk, *Inclusion/exclusion meets measure and conquer*, Proceedings of the 17th Annual European Symposium on Algorithms (ESA 2009), Lecture Notes in Computer Science, vol. 5757, Springer, 2009, pp. 554–565.
- [Wah04] Magnus Wahlström, *Exact algorithms for finding minimum transversals in rank-3 hypergraphs*, Journal of Algorithms **51** (2004), no. 2, 107–121.
- [Wil05] Ryan Williams, *A new algorithm for optimal 2-constraint satisfaction and its implications*, Theoretical Computer Science **348** (2005), no. 2-3, 357–365.
- [XN13] Mingyu Xiao and Hiroshi Nagamochi, *Confining sets and avoiding bottleneck cases: A simple maximum independent set algorithm in degree-3 graphs*, Theoretical Computer Science **469** (2013), 92–104.

(Serge Gaspers) UNSW AUSTRALIA AND NICTA, SYDNEY, AUSTRALIA
E-mail address: `sergeg@cse.unsw.edu.au`

(Gregory B. Sorkin) LONDON SCHOOL OF ECONOMICS, HOUGHTON STREEN, LONDON WC2A 2AE, ENGLAND
E-mail address: `g.b.sorkin@lse.ac.uk`